ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟ ΝΟΣΟΚΟΜΕΙΟ ΤΟΥ 2000

ΓΙΩΡΓΟΣ Δ. ΧΟΥΤΖΟΥΜΗΣ

ΒΑΓΓΕΛΗΣ Γ. ΜΑΝΔΡΑΒΕΛΗΣ

ΕΠΙΒΛΕΠΩΝ
ΚΑΘΗΓΗΤΗΣ κ. ΚΟΥΤΣΟΥΡΗΣ ΔΗΜΗΤΡΙΟΣ

ΑΝΑΓΚΕΣ ΠΕΛΑΤΗ

Πελάτης είναι μια κατασκευαστκή εταιρεία κτιρίων με υψηλό βαθμό αυτοματοποίησης λειτουργιών εσωτερικών και εξωτερικών χώρων. Πελάτης μπορεί να είναι και οποιοσδήποτε ιδιώτης με ανάλογες απαιτήσεις.

Οι ανάγκες του πελάτη είναι:

Η κατασκευή ενός συστήματος βασισμένο σε ηλεκτρονικό υπολογιστήγια έλεγχο και διαχείριση των εσωτερικών χώρων του νοσοκομείου με σκοπό τη μέγιστη δυνατή διευκόλυνση του ιδιοκτήτη.

Ο έλεγχος θα γίνεται στα εξής επίπεδα:

1. Αμεσος έλεγχος των συσκευών χωρίς μεσολάβηση του δικτύου μέσω διακοπτών πάνω στις συσκευές.
2. Αμεσος έλεγχος μέσω κεντρικής κονσόλας (ON-OFF).
3. Χρονοπρογραμματισμός μέσω της κεντρικής κονσόλας
4. Αυτόματη λειτουργία με τη βοήθεια ειδικών αισθητήρων.
5. Τηλεχειρισμός μέσω τηλεφώνου (άμεσος / χρονοπρογραμματιζόμενος)
6. Ελεγχος κατανάλωσης επιλεγμένων συσκευών.

Το σύστημα που θα κατασκευαστεί θα πρέπει να ικανοποιεί τις εξής γενικές απαιτήσεις:

- Ελεγχος των κυριοτέρων συσκευών και συγκεκριμένα:

ΣΥΣΚΕΥΕΣ - ΣΥΣΤΗΜΑΤΑ	ΕΠΙΠΕΔΑ ΕΛΕΓΧΟΥ
Καφετιέρα	1,3,5,6
Απορροφητήρας	1,4,6
Ηλεκτρικός / ηλιακός θερμοσίφωνας	1,2,3,4,5,6
Στερεοφωνικό συγκρότημα	1,3
Συσκευές κλιματισμού	1,3,4,5,6
Σύστημα ασφάλειας – κεντρικό κλείδωμα	1,2,4
Πυρασφάλεια	1,4
Σύστημα αυτόματου ποτίσματος κήπου	1,3,4,6
Φωτεινότητα εσωτερικών χώρων	1,3,4,5,6
Εφεδρικό σύστημα παροχής ενέργειας	1,4
Ενεργοβόρες συσκευές γενικά	1,6

ΠΡΟΔΙΑΓΡΑΦΕΣ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΣΥΣΤΗΜΑ

1_ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Κατασκευή ενός συστήματος βασισμένου σε ηλεκτρονικό υπολογιστή για έλεγχο και διαχείριση των εσωτερικών χώρων ενός νοσοκομείου.

2_ΑΙΤΙΟΛΟΓΗΣΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Η απίτηση ευφυούς, αυτοματου, ταχύ και αξιόπιστου ελέγχου κάποιων εργασιών ενός νοσοκομείου συνεπάγεται τη χρήση υπολογιστικού συστήματος και μάλιστα ενός συστήματος προσωπικού υπολογιστή μαζί με τις απαραίτητες συνδέσεις με τον εξωτερικο φυσικό κόσμο.

3_ΣΚΟΠΟΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΚΑΙ ΤΟΥ ΕΡΓΟΥ

Σκοπός του συστήματος:

- Αύξηση της άνεσης
- Απαλαγγή του Διοικητή του Νοσοκομείου από δουλειές ρουτίνας
- Αύξηση της ασφάλειας
- Κεντρικός έλεγχος της εγκατάστασης
- Συγκέντρωση και αποθήκευση σε βάση δεδομένων στοιχείων από τη λειτουργία των συνιστωσών του συστήματος

Σκοπός του έργου:

Ο σχεδιασμός, η κατασκευή, ο έλεγχος ορθότητας και η εγκατάσταση ενός τέτοιου συστήματος.

4_ΠΕΡΙΟΡΙΣΜΟΙ ΣΤΟ ΣΥΣΤΗΜΑ ΚΑΙ ΤΟ ΕΡΓΟ

Περιορισμοί στο σύστημα:

Ο αλγόριθμος υλοποίησης πρέπει να λάβει υπόψην του το γεγονός ότι οι έλεγχοι όλων των υποσυστημάτων γίνονται σε πραγματικό χρόνο οπότε η καθυστέρηση του λογισμικού και του υλικού θα πρεπει να είναι ελάχιστη δυνατή.

Περιορισμοί στο έργο:

Η παράδοση και η εγκατάσταση του συστήματος να ολοκληρωθεί σε ένα χρόνο.

5_ΛΕΙΤΟΥΡΓΙΕΣ ΑΝΑ ΣΥΝΙΣΤΩΣΑ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

5.1 ΥΛΙΚΟ

Από το υλικό εκτελούνται οι κάτωθι λειτουργίες:

1. Μέσω αισθητήρα θερμοκρασίας περιβάλλοντος
   Μέτρηση της θερμοκρασίας περιβάλλοντος και μεταβίβαση της πληροφορίας στο λογισμικό. Σε περίπτωση αναλογικού οργάνου θα παρεμβάλλεται κατάλληλος μετατροπέας Α/Ψ.
2. Μέσω αισθητήρα φωτεινότητας
   Μέτρηση της φωτεινότητας περιβάλλοντος και μεταβίβαση της πληροφορίας στο λογισμικό. Σε περίπτωση αναλογικού οργάνου θα παρεμβάλλεται κατάλληλος μετατροπέας Α/Ψ.
3. Μέσω αισθητήρα υγρασίας.
   Μέτρηση της υγρασίας περιβάλλοντος και μεταβίβαση της πληροφορίας στο λογισμικό. Σε περίπτωση αναλογικού οργάνου θα παρεμβάλλεται κατάλληλος μετατροπέας Α/Ψ.
4. Μέσω αισθητήρα θερμοκρασίας νερού θερμοσίφωνα
   Μέτρηση της θερμοκρασίας νερού θερμοσίφωνα και μεταβίβαση της πληροφορίας στο λογισμικό. Σε περίπτωση αναλογικού οργάνου θα παρεμβάλλεται κατάλληλος μετατροπέας Α/Ψ.
5. Μέσω αισθητήρα καπνού

Μέσω της συγκεντρωσης καπνού κουζίνας (υποσύστημα ελέγχου απορoφητήρα) καπνού σε όλα τα μέρη του νοσοκομείου (υποσύστημα πυρασφάλειας) και μεταβίβαση της πληροφορίας στο λογισμικό. Σε περίπτωση αναλογικού οργάνου θα παρεμβάλλεται κατάλληλος μετατροπέας Α/Ψ.

6. Μέσω τψν ηλεκτρικών κινητήρων ρύθμισης περσίδων
   Ρύθμιση της κλίσης των περσίδων για μεταβολή της φωτεινότητας
7. Μέσω ηλεκτρικών κινητήρων χειρισμού παραθύρων και εξωτερικών θυρών.
   Ανοιγμα / κλείσιμο παραθύρων εξωτερικών θυρών (σύστημα πυρασφάλειας)
8. Μέσω ροοστατών ρύθμισης έντασης ηλεκτρικών λαμπτήρων
   Ρύθμιση έντασης ηλεκτρικών λαμπτήρων.
9. Μέσω ηλεκτρομαγνητικών κλειδαριών
   Ασφάλιση απασφάλιση παραθύρων, εξωτερικών θυρών.
10. Μέσω συστήματος παρακολούθησης εσωτερικών εξωτερικών χώρων
    Ελεγχος παραβίασης χώρων νοσοκομείου
11. Μέσω συστήματος αυτόματου ποτίσματος
    Συνεργασία με τον αισθητήρα υγρασίας του εδάφους και διαδικασία ποτίσματος.
12. Μέσω γεννήτριας
    Εφεδρική παροχή ενέργειας σε περίπτωση διακοπής ρεύματος
13. Μέσω σταθεροποιητή τάσης για τον Η/Υ
    Σταθεροποίηση τάσης για ομαλή λειτουργία ευαίσθητων συσκευών (κύκλωμα Η/Υ, ηλεκτρονόμοι, αισθητήρες κλπ)
14. Μέσω συστήματος αδιάλειπτης παροχής ρεύματος
    Αδιάλειπτη παροχή ρεύματος
15. Μέσω ηλεκτρομαγνητκών διακοπτών (ηλεκτρονόμων)
    Λειτουργίες ON/OFF συσκευών
16. Μέσω σειρ΄ηνας πυρασφάλειας
    Προειδοποίηση περίπτωσης πυρκαγιάς
17. Μέσω σειρήνας συστήματος ασφαλείας
    Προειδοποίηση παραβίασης
18. Μέσω ψηφιακού τηλεφώνου
    Τηλεχειρισμός επιλεγμένων συσκευών
19. Μέσω μετρητών καταναλωσης
    Καταγραφή κατανάλωσης ενέργειας των επιμέρους συσκευών

5.2 ΛΟΓΙΣΜΙΚΟ

Από το λογισμικό εκτελούνται οι παρακάτω λειτουργίες:

- Φιλικό περιβάλλον κεντρικής κονσόλας
- Λήψη, επεξεργασία δεδομένων από τους αισθητήρες
- Λήψη και επεξεργασία δεδομένων από το τηλέφωνο
- Λήψη πληροφορίας λειτουργίας του UPS με σκοπό την ενεργοποίηση της εφεδρικής γεννήτριας
- Αποστολή σημάτων ελέγχου πρός τις περιφερειακές συσκευές
- Τηρηση ημερολογίου ατζέντας
- Διαχείριση βασεως δεδομένων (εορτολόγιο,στατιστικά στοιχεία)

5.3 ΑΝΘΡΩΠΟΙ

Οι λειτουργίες του συστήματος που εκτελούνται από ανθρώπους είναι:

Α) Από το χρήστη προγραμματιστή μέσω της κεντρικής κονσόλας

- Εντολές αμέσου λειτουργίας συσκευών
- Εντολές χρονοπρογραμματισμού συσκευών
- Εισαγωγή παραμέτρων του συνόλου λειτουργιών
- Εντολές τηλεχειρισμού των συσκευών μέσω τηλεφώνου
- Εισαγωγη στοιχείων στη βάση δεδομένων της ατζέντας
- Αιτήσεις πληροφόρησης για την κατάσταση λειτουργίας του συστήματος στο σύνολό του

Β) Από υπεύθυνο εγκατάστασης συντήρησης

- Επέμβαση σε περίπτωση βλάβης.

6_ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΧΡΗΣΤΩΝ

Στο σύστημα οι χρήστες είναι υπάλληλοι του νοσοκομείου καθώς και οι γιατροί που εργάζονται στο εν λόγω νοσοκομείο. Οι χρήστες χρειάζεται να έχουν μόνο σχετική πείρα χρήσης πληκτρολογίου ή κάποιας άλλης βοηθητικής συσκευής εισόδου δεδομένων στον Η/Υ (ποντίκι)

Κατά την έναρξη λειτουργίας του συστήματος θα πρέπει να υπάρξει κατάλληλη ενημέρωση λειτουργίας, προφανώς μόνο εντελώς επιφανειακά στο επίπεδο του χρήστη. Δε χρειάζεται να έχουν οποιαδήποτε άλλη τεχνική επιδεξιότητα ή εκπαίδευση.

Φυσικά το σύστημα θα περέχει κάθε απαραίτητη πληροφορία χειρισμού του συστήματος – on line help.

7_ΠΕΡΙΒΑΛΛΟΝΤΑ

7.1 ΑΝΑΠΤΥΞΗΣ

Η εξομοίωση της πραγματικής λειτουργίας του συστήματος θα γίνει σε προσωπικό υπολογιστή δλδ με μικροεπεξεργαστή 80486. Το λειτουργικό σύστημα που θα χρησιμοποιηθει είναι το MS DOS v. 6.3

Το λογισμικό θα μεταγλωτιστεί μεσω του TURBO PASCAL COMPILER v.6.

7.2 ΛΕΙΤΟΥΡΓΙΑΣ

Το πραγματικό περιβάλλον που θα υλοποιηθεί το σύστημα θα είναι προσωπικός υπολογιστής που θα συνδέεται μέσω διαπροσωπειών με τους αισθητήρες, ηλεκτρονόμους, ανιχνευτές, τηλέφωνο, ηλεκτρομαγνητικές κλειδαριές.

Το λειτουργικό σύστημα που θα χρησιμοποιηθεί θα είνια το MS DOS v6.3 Το λογισμικό θα μεταγλωτιστεί μέσω του TURBO PASCAL COMPILER v.6 σε γλώσσα μηχανής του μικροεπεξεργαστή 80486.

7.3 ΣΥΝΤΗΡΗΣΗΣ

Ιδιο με λειτουργίας.

8_ΣΤΡΑΤΗΓΙΚΗ ΛΥΣΗΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Στη διαδικασία επίλυσης του προβλήματος θα ακολουθηθούν τα επόμενα βήματα:

- Προσδιορισμός των απαιτήσεων από το συστημα
- Προσδιορισμός των απαιτήσεων από το λογισμικό
- Προσδιορισμός των απαιτήσεων από το υλικό
- Σχεδίαση λογισμικού
- Κωδικοποίηση και εκκαθάριση λαθών
- Ελεγχος ορθότητας του λογισμικού και εξομοίωση
- Σχεδίαση των εξωτερικών διαπροσωποιών επικοινωνίας του λογισμικού με τα ηλεκτρομηχανικά μέρη
- Ελεγχος του συστήματος με φανταστικά δεδομένα
- Ελεγχος του συστήματος με πραγματικά δεδομένα

9_ΠΡΟΤΕΡΑΙΟΤΗΤΕΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Στο σύστημα δίνεται προτεραιότητα στα εξής χαρακτηριστικά:

- Δυνατότητα συνεχούς αδιάλειπτης λειτουργίας
- Δυνατότητα τοπικου ελέγχου των συσκευών παρακάμπτοντας τον Η/Υ σε περίπτωση λάθους
- Εμφαση στο φιλικό και κατανοητό περιβάλλον χειρισμού

10_ΚΡΙΤΗΡΙΑ ΑΠΟΔΟΧΗΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Καλη σχεση προσφοράς υπηρεσιών προς κόστος

11_ΠΗΓΕΣ ΠΛΗΡΟΦΟΡΙΩΝ

Ντοκυμαντερ και άρθρα για τις τάσεις και διαμορφώσεις των μελλοντικων νοσοκομείων.

ΠΛΑΝΟ ΤΟΥ ΕΡΓΟΥ

1_ΕΙΣΑΓΩΓΗ

Στο πλάνο του έργου φαίνονται οι εκτιμήσεις, σχέσεις και χρονικές εξαρτήσεις μεταξύ παραδοτέων αντικειμένων και η κατανομή πόρων για την παραγωγη των παραδοτέων αντικειμένων. Το πλάνο πρέπει να συνοδεύεται από το οργανόγραμμα του έργου και από ένα καθορισμό των διαδικασιών και προτ΄υπων που θα χρησιμοποιηθούν κατά την κατασκευή. Είναι σημαντική η ύπαρξη του

εγγράφου αυτού για την ολοκληρωμένη και μέσα στα χρονικά όρια παράδοση του έργου.

2_ΠΟΡΟΙ

2.1 ΑΝΘΡΩΠΟΙ

Η υλοποίηση της όλης εφαρμογής έγινε από δύο φοιτητές του Ε.Μ. Πολυτεχνείου του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Η γνώση και η εμπειρία που αποκτήθηκαν στα προηγούμενα έτη σπουδών ήταν απαραίτητα και καθοριστικά στοιχεία για την ανάπτυξη της εφαρμογής. Γνώσεις προγραμματισμού και προχωρημένων αρχών αυτού καθώς και εξοικείωση με γνωστά περιβάλλοντα προγραμματισμού ήταν απαραίτητα για τη σωστή ολοκλήρωση του έργου.

2.1 ΜΗΧΑΝΕΣ

Η συγκεκριμένη εφαρμογή έχει αναπτυχθεί σε μηχανές IBM AT και συμβατές αυτών. Το ελάχιστο υλικό που απαιτείται είναι ενα IBM συμβατό μηχάνημα με 4 MB RAM, ένα floppy drives, ένα σκληρό δίσκο, μία κάρτα γραφικών σειριακή και παράλληλη θύρα επικοινωνίας για τη διασύνδεση των εξωτερικών συσκευών.

2.3 ΕΡΓΑΛΕΙΑ

Τα εργαλεία που χρησιμοποιούνται για την ανάπτυξη της εφαρμογής είναι:

- Λειτουργικό σύστημα MS DOS από την έκδοση 6.2
- Το ολοκληρωμένο περιβάλλον της TURBO PASCAL 6.0 που περιλαμβάνει εργαλεία όπως:
  - Editor
  - Compiler
  - File managing
  - Χρήσιμες βιβλιοθήκες (TURBO VISION)
  - Debugging κ.α
- Επίσης για τη δημιουργία των εγγράφων χρησιμοποιήθηκε ο επεξεργαστής κειμένου Word

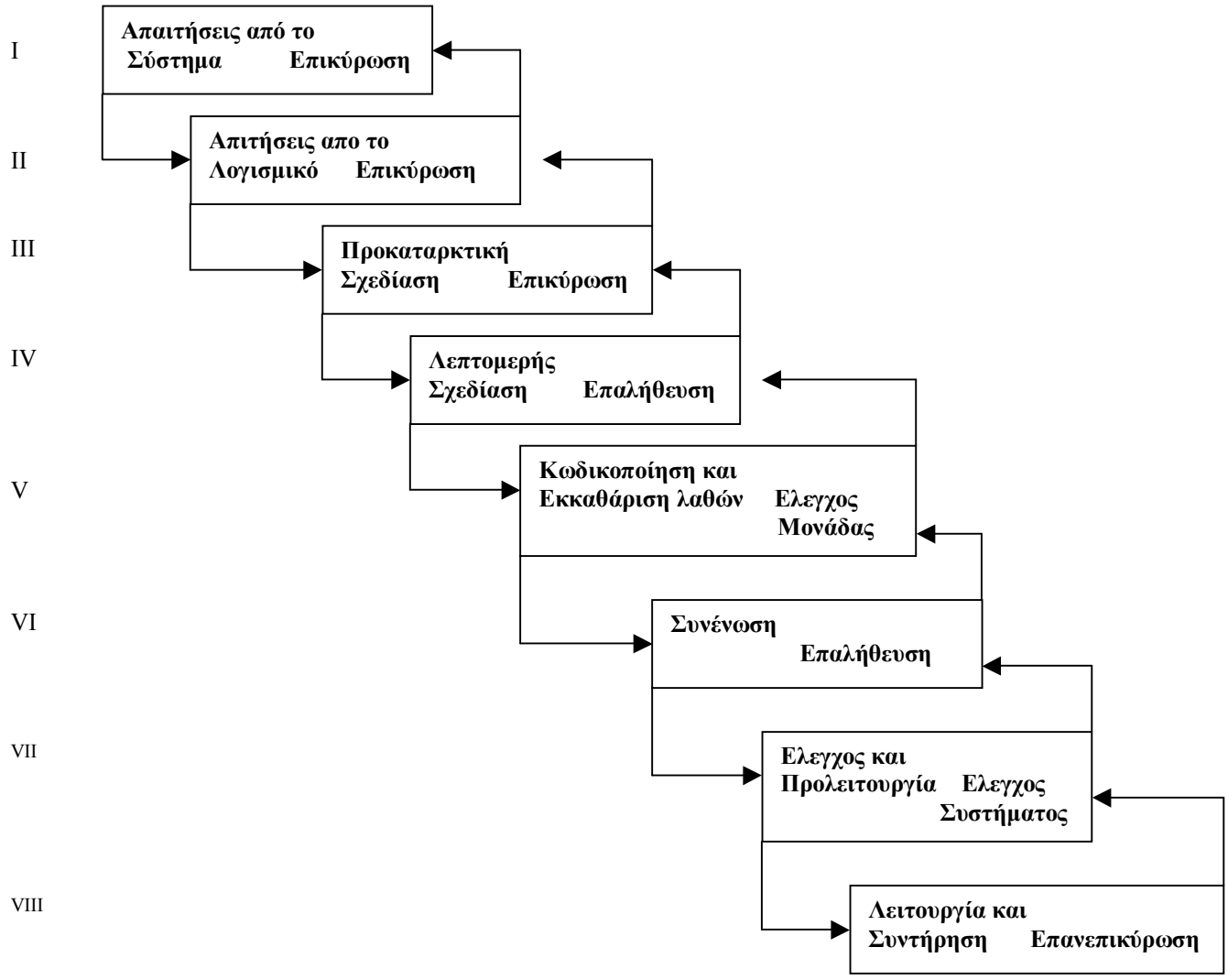
3. ΜΕΘΟΔΟΛΟΓΙΑ

3.1 ΜΟΝΤΕΛΟ ΚΥΚΛΟΥ ΖΩΗΣ

Η κατασκευή της εφαρμογής ακολούθησε το μοντέλο του καταράκτη το οποίο αποτελείται απο 8 στάδια αναπτυξης λογισμικού. Εκτελούμε τις φάσεις – στάδια ακολουθιακα. Κάθε φορά που τελειώνουμε ένα στάδιο και το διορθώνουμε ελέγχουμε όσο το δυνατόν καλύτερα για την ανεύρεση λαθών και το διορθώνουμε. Είναι πολύ

σημαντικός ο καλός έλεγχος γιατί κάθε φορά το επόμενο στάδιο βασίζεται στο προηγούμενο και άρα η ορθότητά του στην ορθότητα του προηγούμενου σταδίου. Ανακάλυψη λαθών εκ των υστέρων συνεπάγεται μεγάλες οπισθοδρομήσεις και άρα αύξηση του χρόνου και του κόστους υλοποίησης. Μειονέκτημα της μεθόδου είναι ότι δεν διατηρεί απικοινωνία με τα αρχικά στάδια όπως οι απαίτήσεις του πελάτη και οι απαιτήσεις του λογισμικού. Θα πρέπει όμως καθ'όλη τη σχεδίαση και εκτέλεση του έργου να είναι ξεκάθαροι οι αρχικοί στόχοι και απαιτήσεις.
Αυτό έχει ληφθεί υπ'όψην κατά τη σχρδίαση της παρούσης εφαρμογής για να αντιμετωπιστεί εν μέρει το μειονέκτημα του μοντέλου του καταράκτη.

Ενα διάγραμμα του μοντέλου φαίνεται παρακάτω:



3.2 ΠΡΟΣΕΓΓΙΣΗ ΣΧΕΔΙΑΣΗΣ

Ως γνωστόν πολύ σημαντικό για ένα έργο είναι η ύπαρξη σχεδίου εξαρχής και η ακολούθησή του. Χωρίς την ύπαρξη σχεδίου είναι πολύ εύκολο να αποτύχει ολόκληρο το έργοή το αποτέλεσμα να μήν είναι αποδοτικό όσο θα έπρεπε να είναι. Η σχεδίαση είναι σημαντική και δημιουργική εργασία πού όμως δε μπορεί να γίνει με αυστηρά αλγοριθμικό τρόπο. Απλά μπορεί να ακολουθηθούν γενικές κατευθυντήριες

γραμμές και γενικές οδηγίες. Ο σχρδιαστής χρησιμοποιεί την εμπειρία του τη διαίσθησή του και τις δημιουργικές του ικανότητες για να ακολουθήσει ένα συγκεκριμένο τρόπο σχεδίασης.

Το σημείο εκκίνησης της σχεδίασης είναι το πρώτο ερωτηματικό που αντιμετωπίζεται από το σχεδιαστή. Εδώ στη συγκεκριμένη εφαρμογή χρησιμοποιείται η λεγόμενη «από άνω προς τα κάτω στρατηγική» που σαν σημείο εκκίνησης θεωρεί την αποψη του όλου συστήματος σαν οντότητας και στη συνέχεια αναλύεται – χωρίζεται σε υποσυστήματα και αλλες πιο αναλυτικές δομικές μονάδες.

Σαν τεχνική χρησιμοποιείται η δομημένη σχεδίαση σε συνδιασμό με το δομημένο προγραμματισμό και τη δομημένη ανάλυση. Είναι πολύ δημοφιλής τεχνική και χρησιμοποιείται ευρέως.

3..3 ΠΡΟΣΕΓΓΙΣΗ ΚΩΔΙΚΟΠΟΙΗΣΗΣ

Για την παραγωγή κώδικα οι προγραμματιστές χρησιμοποιούν τα λεπτομερή σχέδια μονάδων του συστήματος έχοντας όμως υπ'όψην τους και τις προδιαγραφές των απαιτήσεων από το σύστημα, από το λογισμικό και το σχέδιο λογισμικού. Επίσης κατά την κωδικοποίηση απαιτείται η γνώση της ταυτότητας της γλώσσας και των τεχνοτροπιών προγραμματισμού. Χρειάζεται η καλή γνώση και η πείρα της συγκεκριμένης γλώσσας υλοποίησης αλλά και των γενικών αρχών προγραμματισμού.

3.4 ΠΡΟΤΥΠΑ

Τα πρότυπα που χρησιμοποιήθηκαν ξκατά τη διάρκεια της ανάπτυξης της εφαρμογής είναι του IEEE που τα εχει αποδεχτεί και το ANSI (American National Standard System)

4. ΚΟΣΤΟΣ

Η εκτίμηση του κόστους του λογισμικού είναι από τα δυσκολότερα προβληματα. Η προσπάθεια της εκτίμησης του κόστους από τη φάση σχεδίασης είναι επιρεπής σε λάθη γιατί υπάρχει πλήθος αγνώστων και αστάθμητων παραγόντων που μπορεί να επηρεάσουν καθορισστικά το κόστος του έργου.

Στην προσπάθεια μας να κοστολογήσουμε αυτό το έργο χρησιμοποιούμε μη αλγοριθμική τεχνική του κατακερματισμού και μάλιστα τη δομή εκείνη που θεωρεί ως κατακερματιζόμενο σύστημα την όλη διεργασία κατασκευής του έργου. Ετσι η διεργασία κατασκευής του έργου μπορεί να χωριστεί σύμφωνα με το σχέδιο.

ΔΙΕΡΓΑΣΙΑ

Υποδιεργασία 1: Σύνταξη των απαιτήσεων από τον πελάτη και από το σύστημα (5%)
Υποδιεργασία 2: Σύνταξη των απαιτήσεων από το λογισμικό (5%)
Υποδιεργασία 3: Κατασκευή σχεδίου (23%)
Υποδιεργασία 4: Κωδικοποίηση μινάδων λειτουργικής απαίτησης menu(12%)
Υποδιεργασία 5: Κωδικοποίηση μονάδων λειτουργικής απαίτησης polling (7%)
Υποδιεργασία 6: Κωδικοποίηση μονάδων λειτουργικής απάιτησης 3 (4%)
.
.

.
Υποδιεργασία 18: Κωδικοποίηση μονάδων λειτουργικής απαίτησης 15 (4%)
Υποδιεργασία 19: Συνένωση, έλεγχος, επικύρωση και τελική παρουσίαση προϊόντος (10%)

Αν αναλογιστούμε τώρα ότι μικρότερη διεργασία (4%) μπορεί να πάρει και δύο εργασιμες μέρες και αυτές να αμείβονται με το ποσό των 10.000 δρχ η κάθε μία τότε το τελικό κόστος του έργου θα είναι (100/4)*10000*2 =500.000 δρχ. Το κόστος αυτό είναι μάλλον μία αβέβαιη εκτίμηση που έγινε κατά τη φάση της σχεδίασης.

5_ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Οπως και με την κοστολόγηση το έργο χωρίζεται σε επιμέρους διαδικασίες και υπολογίζεται η διάρκεια της καθε μίας χωριστά. Υστερα βρίσκονται οι διαδικασίες που μπορούν να γίνουν παράλληλα και έτσι καταλήγουμε στο χρονοδιάγραμμα. Ενα μέτρο καταμερισμού του χρόνου εργασίας είναι οτι οι απαιτήσεις και η σχεδίαση παίρνουν διπλάσιο χρόνο από την κωδικοποίηση. Επίσης η διάρκεια των επιμέρους εργασιών συνηθίζεται να μετριέται σε μήνες προγραμματιστή.

Το έργο μπορεί να χωριστεί σε τρείς υποενότητες οι οποίες μπορεί να δουλεύονται παράλληλα από τους 3 προγραμματιστες της ομάδας.
Το χρονοδιάγραμμα για την τρέχουσα εφαρμογή φαίνεται παρακάτω:

Εγγραφα - διεργασίες	**Ηερομηνία**
Ανάγκες πελάτη Προδιαγραφέα απαιτήσεων από το σύστημα	5-10-1996
Προδιαγραφές απαιτήσεων από το λογισμικό	30-10-1996
Περιγραφή σχεδίου λογισμικού	5-12-1996
Τεκμηρίωση δυναμικού ελέγχου	20-01-1997
Ολοκληρωμένη παράδοση του έργου	27-03-1997

ΠΡΟΔΙΑΓΡΑΦΕΣ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ

1_ΕΙΣΑΓΩΓΗ

1.1 ΣΚΟΠΟΣ

Σκοπός του εγγράφου προσδιορισμού των απαιτήσεων από το λογισμικο είναι να περιγράψει τις λειτουργίες του συστήματος ελέγχου και διαχείρισης των εσωτερικών εξωτερικών χώρων του νοσοκομείου οι οποίες εκτελούνται από το λογισμικό, καθώς και τους περιορισμούς στους οποίους θα υπόκεινται αυτές οι λειτουργίες.

Αυτό το έγγραφο θα χρησιμοποιηθεί ως βάση για τη σχεδίαση κωδικοποίηση έλεγχο και συντήρηση του λογισμικού.

Το έγγραφο αυτό θα χρησιμοποιηθει :

- Σαν σημείο αναφοράς από τους σχεδιαστές λογισμικού και
- Σαν περιγραφή των λειτουργιών του συστήματος που θα εκτελούντια απο το λογισμικό

1.2 ΓΕΝΙΚΗ ΕΚΤΑΣΗ

Τα γενικά χαρακτηριστικά του λογισμικού είναι

- Κεντρική διαχείριση των χώρων του νοσοκομείου χωρίς να αποκλείεται ο επι τόπου χειρισμός τψν συσκευών
- Λήψη δεδομένων απο τα υποσυστήματα που περιλαμβάνουν αισθητήρες και επεξεργασία τους απο το λογισμικό
- Αποστολή σημάτων κατάλληλων ανάλογα με τα δεδομένατων αισθητήρων και τις εντολές του χρήστη μέσω της κεντρικής κονσόλας (άμεσες, χρονοπρογραμματιζόμενες

Διεργασίες και ταυτότητες

Κάθε διεργασία στο Unix έχει μία "ταυτότητα" - έναν αύξων αριθμό, δηλαδή, που της δίνει το λειτουργικό σύστημα. Μπορείτε να ανακαλύψετε την ταυτότητα που έχει το πρόγραμμά σας όταν εκτελείται (όταν, δηλαδή, αποτελεί διεργασία) χρησιμοποιόντας την κλήση **getpid**():

```
#include <unistd.h>

pid_t myid;

myid = getpid();
printf("My pid is %d", myid);
```

Δημιουργία νέων διεργασιών - fork()

Η βασική ρουτίνα που επιτρέπει σε μία διεργασία να δημιουργήσει δυναμικά άλλες διεργασίες στο Unix είναι η **fork**(). Η κλήση αυτή δημιουργεί ένα αντίτυπο της διεργασίας που την καλεί. Θα πρέπει να φαίνεται σχετικά ανώφελο να δημιουργήσουμε αντίτυπα της ίδιας διεργασίας, από τη στιγμή που συνήθως θέλουμε η κάθε διεργασία να κάνει κάτι διαφορετικό. Αυτός, όμως, είναι ο μόνος τρόπος. Παρ' όλα αυτά, από τη στιγμή που θα δημιουργηθούν τα αντίτυπα μπορούν να 'διαφοροποιηθούν' μεταξύ τους εξετάζοντας την τιμή που επέστρεψε η **fork**().
Η λειτουργία της **fork**() γίνεται πιο κατανοητή μέσα από ένα απλό παράδειγμα. Ας πάρουμε τον παρακάτω κώδικα:

```
    #include <unistd.h>


    ...
(α)  x = fork();
(β)  if (x == 0)
(γ)    < κώδικας A >
(δ)  else
(ε)    < κώδικας B >
    ...
```

Το πρόγραμμα (διεργασία) εκτελείται κανονικά μέχρι τη γραμμή (α), όπου γίνεται η κλήση στη συνάρτηση **fork**(). Η κλήση '**fork**()' έχει σαν αποτέλεσμα τη δημιουργία ενός νέου αντιτύπου της καλούσας διεργασίας το οποίο αρχίζει να εκτελείται μαζί

της. Όπως είπαμε, αυτός είναι ο μόνος τρόπος να δημιουργηθούν νέες διεργασίες: όχι διαφορετικές, αλλά αντίγραφα της ίδιας διεργασίας, η οποία ονομάζεται διεργασία γονέας (parent process). Το αντίγραφο ονομάζεται διεργασία παιδί (child process) και εκτελείται ακριβώς μετά το σημείο που έγινε η κλήση στην **fork**(). Με άλλα λόγια, αφού δημιουργηθεί η διεργασία-παιδί, η εκτέλεσή της ξεκινά από το σημείο (α), σαν να έχει επιστρέψει από την **fork**() - δεν ξεκινά από την αρχή (δηλαδή από την main() της).
Στο παράδειγμά μας, τόσο η διεργασία γονέας όσο και το νεοδημιουργημένο παιδί επιστρέφουν μαζί από την **fork**() στη γραμμή (α). Το παιδί, δηλαδή, δεν εκτελεί τον κώδικα που υπήρχε πριν.
Κατά την δημιουργία του αντιγράφου, ΑΝΤΙΓΡΑΦΟΝΤΑΙ ΤΑ ΠΑΝΤΑ: και ο κώδικας, αλλά και οι μεταβλητές. Επομένως, οι δύο (πλέον) διεργασίες έχουν τον δικό τους κώδικα και τις δικές τους (ιδιωτικές) μεταβλητές οι οποίες όμως θα έχουν αρχικά ακριβώς τις ίδιες τιμές.
Η κλήση στην **fork**() επιστρέφει:

- στον μεν γονέα, την ταυτότητα της διεργασίας-παιδί
- στο δε παιδί, την τιμή 0

Επομένως, τα αντίγραφα της διεργασίας μπορούν να διαφοροποιηθούν μεταξύ τους με βάση την τιμή που επιστρέφει στο καθένα από αυτά η **fork**(). Έτσι, στον παραπάνω κώδικα, κατά την έναρξη εκτέλεσης του παιδιού, η μεταβλητή x θα έχει την τιμή 0, και επομένως το παιδί θα εκτελέσει τον κώδικα Α.
Αντίθετα, στον πατέρα το x θα έχει μία τιμή διάφορη του μηδενός (η οποία, όπως είπαμε, είναι η ταυτότητα
του παιδιού) και επομένως δεν θα επαληθευτεί η συνθήκη (β), οπότε θα μεταφερθεί στην γραμμή (δ) και θα
εκτελέσει τον κώδικα Β.
Σημειώστε ότι το παιδί μπορεί να ανακαλύψει την ταυτότητα του γονέα του χρησιμοποιώντας την κλήση **getppid**().

Οικογενειακές σχέσεις (γονέας - παιδί)

Όταν μία διεργασία δημιουργεί κάποιες διεργασίες-παιδιά θα πρέπει συνήθως, πριν ολοκληρώσει την εκτέλεσή της,
να περιμένει να τερματίσουν όλα τα παιδιά της. Αν δεν το κάνει αυτό, τότε τα παιδιά, μόλις τελειώσουν
(π.χ. κάνουν exit()), γίνονται "**zombie**". Με άλλα λόγια, δεν εξαφανίζονται εντελώς από το σύστημα και σπαταλούν κάποιους πόρους. Για τον λόγο αυτό, θα πρέπει κάθε διεργασία-γονέας να εκτελέσει την κλήση **wait**() για κάθε ένα από τα παιδιά που δημιούργησε. Αν δημιούργησε N παιδιά, τότε μπορεί να κάνει:

```
#include <sys/types.h>
#include <sys/wait.h>

for (i = 0; i < N; i++)
  wait(0);
```

Η **wait**(0) σταματά την εκτέλεση του γονέα μέχρι να τερματίσει (όποτε γίνει αυτό) *οποιοδήποτε* από τα παιδιά του. Αν
ενδιαφερόμαστε να περιμένουμε τον τερματισμό ενός συγκεκριμένου παιδιού, μπορούμε να χρησιμοποιήσουμε, αντί της **wait**(), την κλήση **waitpid**() η οποία παίρνει τρεις παραμέτρους (δείτε τα man pages).

Κοινή μνήμη

Για να προγραμματίσετε χρησιμοποιώντας κοινή μνήμη μεταξύ διεργασιών, απαιτούνται χονδρικά τα παρακάτω:

1. *Πριν δημιουργηθούν οι διεργασίες-παιδιά*, ο γονέας θα πρέπει να ζητήσει δυναμικά κάποιο χώρο στην μνήμη, ο οποίος θα χρησιμοποιηθεί για την αποθήκευση των κοινών δεδομένων.
2. Ο χώρος αυτός θα πρέπει να '*προσαρτηθεί*' στο χώρο διευθύνσεων του γονέα.
3. Οι διεργασίες μπορούν, κατόπιν, να δημιουργηθούν από το γονέα και όλες θα *κληρονομήσουν* τον κοινό χώρο στη μνήμη.

Προκειμένου να γίνουν όλα τα παραπάνω, κάθε πρόγραμμα θα πρέπει να περιλαμβάνει τα παρακάτω αρχεία:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

Οι βασικές κλήσεις που χρειάζεται κανείς είναι οι **shmget**(), **shmat**() και **shmctl**(). Η δέσμευση της κοινής μνήμης γίνεται με την συνάρτηση **shmget**(). Δεν θέλουμε να μπούμε στις λεπτομέρειες των παραμέτρων της **shmget**() και για αυτό δίνουμε το πιο συνηθισμένο τρόπο κλήσης της:

```
int memid = shmget(IPC_PRIVATE, size, 0600|IPC_CREAT);
```

όπου size είναι ο αριθμός των bytes που θέλουμε να δεσμεύσουμε και memid είναι μία 'ταυτότητα' για το τμήμα της μνήμης που θα δοθεί. Την ταυτότητα αυτή μπορούμε να τη χρησιμοποιήσουμε για την προσάρτηση την μνήμης στην διεργασία μας, ως εξής:

```
void *p = shmat(memid, 0, 0);
```

Αυτό που επιστρέφει η **shmat**() είναι ένας δείκτης (pointer) στην αρχή της κοινής μνήμης. Αυτόν το δείκτη μπορεί να τον χρησιμοποιήσει κάθε διεργασία που θα δημιουργήσουμε για να τοποθετεί και να τροποποιεί (κοινά) δεδομένα. Έτσι, αν π.χ. θέλουμε να χρησιμοποιήσουμε σαν κοινές μεταβλητές δύο ακεραίους, θα απαιτηθούν ενέργειες της μορφής:

```
*((int *) p) = 0;
*(((int *) p) + 1) = *((int *) p);
```

Στην πρώτη γραμμή αρχικοποιείται ο πρώτος ακέραιος (4 πρώτα bytes στην κοινή μνήμη) στο 0. Στη δεύτερη γραμμή, καταχωρούμε στον δεύτερο ακέραιο την τιμή του πρώτου. Οι εμπλεκόμενοι χειρισμοί είναι αρκετά δύσκολοι και απαιτούν προσοχή στον χειρισμό των δεικτών. Ένα απλό τρικ για να απλοποιηθεί ο χειρισμός του κοινού χώρου είναι το εξής: αν στο πρόγραμμά μας επιθυμούσαμε π.χ. τις δύο παρακάνω (ακέραιες) κοινές μεταβλητές, συν κάποια κοινή float
μεταβλητή, τότε μπορούμε να δηλώσουμε την ακόλουθη δομή στον πρόγραμμά μας:

```
struct whatever
{
  int i, j;
  float k;
} *myvars;
```

και καλούμε την **shmat**() ως εξής:

```
myvars = (struct whatever *) shmat(memid, 0, 0);
```

Από εκεί και μετά, μπορούμε να χειριστούμε σχετικά εύκολα τις κοινές μεταβλητές όπως:

```
myvars->k = 0.0;
myvars->j = myvars->i;
```

χωρίς να μπλεκόμαστε με πολύπλοκους χειρισμούς δεικτών!
Τέλος, θα πρέπει να τονιστεί ότι το τμήμα της κοινής μνήμης που δεσμεύει κάποιο πρόγραμμα, εξακολουθεί να είναι δεσμευμένο ακόμα και μετά την ολοκλήρωση της εκτέλεσης. Είναι λοιπόν απαραίτητο η διεργασία-γονέας, πριν τερματίσει, να καταστρέφει ότι έχει δεσμεύσει, χρησιμοποιώντας την κλήση **shmctl**():

```
shmctl(memid, IPC_RMID, 0);
```

όπου memid είναι η ταυτότητα που επέστρεψε η **shmget**(). Αν ξεχαστείτε, μπορείτε (κατόπιν εορτής) να απελευθερώσετε την κοινή μνήμη χρησιμοποιώντας τα εργαλεία **ipcs** και **ipcrm** (δείτε τα αντίστοιχα manual pages).

Σημαφόροι (semaphores)

Ο βασικός μηχανισμός αμοιβαίου αποκλεισμού που παρέχει το Unix είναι οι σημαφόροι, οι οποίοι είναι αρκετά πολύπλοκοι, με πληθώρα επιλογών στη χρήση τους. Θα δούμε εδώ, χωρίς πολλές επεξηγήσεις, ποιος είναι ο συνηθέστερος τρόπος εφαρμογής τους. Κάθε πρόγραμμα θα πρέπει να περιλαμβάνει τα παρακάτω αρχεία:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

και οι βασικότερες κλησεις είναι οι **semget**(), **semctl**() και **semop**(). Η δημιουργία σημαφόρων γίνεται με την συνάρτηση **semget**(), η οποία επιστρέφει μία 'ταυτότητα' που τους προσδιορίζει μοναδικά:

```
int semid = semget(IPC_PRIVATE, N, 0600|IPC_CREAT);
```

όπου δεύτερη παράμετρος προσδιορίζει το *πλήθος* των σημαφόρων που πρέπει να δημιουργηθούν. Η αρχικοποίηση του i-οστού σημαφόρου σε μία τιμή k γίνεται με τον εξής περίεργο τρόπο:

```
union semun {  int              val;      /* Πρέπει να το ορίσουμε εμείς! */
               struct semid_ds *buf;
               ushort_t        *array;
             } arg;

arg.val = k;
semctl(semid, i, SETVAL, arg);
```

Αυτός είναι ο γενικότερος τρόπος, αν και σε μερικά συστήματα μπορεί κανείς πιο απλά να γράψει:

```
shmctl(semid, i, SETVAL, k);
```

Οι λειτουργίες up() και down() που περιγράφονται στο βιβλίο σας υλοποιούνται μέσω κλήσεων στην **semop**(), ως εξής.

```
struct sembuf operation;

semop(semid, &operation, 1);
```

Η δομή struct sembuf έχει τρία πεδία: sem_num, sem_op και sem_flg. Το sem_flg το θέτουμε συνήθως στην
τιμή 0, το sem_num προσδιορίζει σε ποιον από τους N σημαφόρους αναφερόμαστε (βλ. παραπάνω) ενώ το sem_op πρέπει να είναι -1 αν ενδιαφερόμαστε για την λειτουργία της down(), και +1 αν ενδιαφερόμαστε για την up(). Π.χ., στην πρώτη περίπτωση, η ολοκληρωμένη λειτουργία για τον i-οστό σημαφόρο έχει ως εξής:

```
struct sembuf operation;

operation.sem_flg  = 0;
operation.sem_num  = i;
operation.sem_op   = -1;

semop(semid, &operation, 1);          /* Down */
```

Τέλος, πριν τερματίσει ένα πρόγραμμα που κάνει χρήση σηματοφόρων, θα πρέπει να αποδεσμεύσει την μνήμη που καταλαμβάνουν, ως εξής:

```
semctl(semid, 0, IPC_RMID);
```

όπου καταστρέφονται και οι N σημαφόροι.

Η πρώτη ερώτηση είναι *τι θέλουμε να χρονομετρήσουμε;* Ενδιαφερόμαστε για *πραγματικό χρόνο* ή για *καθαρό χρόνο εκτέλεσης* (δηλαδή χρόνο που αφιέρωσε το σύστημα μόνο για το δικό μας πρόγραμμα - το λεγόμενο *CPU time*); Ο πραγματικός χρόνος είναι ο χρόνος μέχρι το πέρας του προγράμματος και εξαρτάται από το φόρτο του συστήματος. Όσο πιο φορτωμένο είναι το σύστημα, τόσο πιο αργά διεκπεραιώνονται οι διεργασίες. Ο καθαρός χρόνος εκτέλεσης από την άλλη δεν εξαρτάται από τον φόρτο γιατί π.χ. αν η CPU πρέπει να εκτελέσει 10 εντολές του 1sec η κάθε μία, θα αφιερώσει χρόνο 10sec συνολικά άσχετα αν λόγω φόρτου εκτελεί τις εντολές μας μία-μία ανά ώρα.

Υπάρχουν αρκετοί τρόποι για χρονομέτρηση στο Unix. Ομαδοποιούνται, όμως, σε δύο κύριες κατηγορίες. Η πρώτη χρησιμοποιεί το/α ρολόϊ/α του συστήματος για να μετρήσει κύκλους ρολογιού σε πραγματικό χρόνο (cycle-based, χρειάζεται βοήθεια από το hardware) και η δεύτερη κάνει τις μετρήσεις της ανά τακτά χρονικά διαστήματα ενώ μπορεί να μετρήσει και καθαρούς χρόνους εκτέλεσης (interval-based, το διάστημα καθορίζεται από το λειτουργικό σύστημα).

Στην πρώτη κατηγορία θα βρείτε συνήθως τρόπους οι οποίοι είτε δεν είναι πάντα διαθέσιμοι είτε διαφέρουν από σύστημα σε σύστημα (χρειάζεται πιθανώς και assembly για να έχετε προσπέλαση στους timers) και οι οποίοι βέβαια δεν χάνουν ούτε κύκλο σε ακρίβεια. Σε αυτή την κατηγορία μπορεί να καταταχθεί και η **gettimeofday()**, αν και σε μερικά συστήματα υλοποιείται όπως οι μέθοδοι της δεύτερης κατηγορίας οπότε πιθανά δεν έχει μεγάλη

ακρίβεια. Το βασικό μειονέκτημα αυτών των μεθόδων είναι ότι οι μετρήσεις δίνουν τον *πραγματικό χρόνο* που πέρασε, πράγμα που σημαίνει ότι θα δώσουν διαφορετικές μετρήσεις όταν το σύστημα είναι ελάχιστα φορτωμένο απότι όταν το σύστημα είναι πολύ φορτωμένο (διότι μέχρι να ολοκληρωθεί η διεργασία που μετράμε θα έχουν συμβεί πολλά context switches και θα έχουν χρησιμοποιήσει την CPU άλλες διεργασίες).

Στην δεύτερη κατηγορία έχουμε συναρτήσεις οι οποίες διαθέτουν σχετικά μικρή ακρίβεια - σπάνια θα δείτε ακρίβεια καλύτερη από ένα εκατοστό του δευτερολέπτου. Όμως, αυτές οι συναρτήσεις μπορούν να υπολογίσουν τον χρόνο που χρησιμοποιήθηκε η CPU μόνο από την υπό μέτρηση διεργασία και άρα δεν επηρεάζονται (πολύ) από τον συνολικό φόρτο του συστήματος. Συναρτήσεις αυτής της κατηγορίας είναι οι **times()** και **clock()**.

Γενικός χοντροκομμένος κανόνας:

- Αν αυτό που θα μετρήσετε παίρνει χρόνο τάξεως κλάσματος του δευτερολέπτου, πρέπει να χρησιμοποιήσετε συναρτήσεις της πρώτης κατηγορίας, να φροντίσετε το σύστημα να μην είναι φορτωμένο και να προσέξετε πολύ στην χρήση των συναρτήσεων για να έχετε την μεγάλη ακρίβεια που απαιτείται.
- Αν αυτό που θα μετρήσετε χρειάζεται δευτερόλεπτο και πάνω, χρησιμοποιείστε συναρτήσεις της δεύτερης κατηγορίας, με τις οποίες θα ασχοληθούμε εδώ. Η ακριβειά τους είναι υπεραρκετή και ο φόρτος του συστήματος δεν της προκαλεί σοβαρά προβλήματα. Η **times()** μπορεί να μερτήσει τόσο πραγματικό χρόνο όσο και καθαρό χρόνο εκτέλεσης. Η **clock()** μετρά μόνο καθαρό χρόνο εκτέλεσης (CPU time).

2. Στην πράξη (I) - η clock() για μέτρηση CPU time

Ένας τρόπος να χρονομετρήσετε (με ακρίβεια συνήθως εκατοστού του δευτερολέπτου) το CPU time των προγραμμάτων σας στο Unix, είναι να κάνετε χρήση της συνάρτησης **clock()**. Θα πρέπει να συμβουλευτείτε τα manual pages για να μάθετε περισσότερα, αλλά εδώ σας δίνονται σχεδόν όλα όσα θα χρειαστείτε.
Για να χρησιμοποιήσετε την **clock()** θα πρέπει να κάνετε **#include <time.h>** στο πρόγραμμά σας. Η **clock()** επιστρέφει τον χρόνο που αφιέρωσε η CPU από τη στιγμή που ξεκίνησε το πρόγραμμά σας. Αν επομένως βάλετε την **clock()** αμέσως πριν και αμέσως μετά το σημείο **A** στο πρόγραμμά σας:

```
t1 = clock();
<κώδικας A>
t2 = clock();
```

η διαφορά των τιμών που θα πάρετε (t2 - t1) σας δίνει ακριβώς πόσο CPU time χρειάστηκε το **A** για να εκτελεστεί. Ο χρόνος αυτός όμως είναι ένας ακέραιος (συνήθως) ο οποίος δεν είναι σε δευτερόλεπτα. Για να βρείτε το χρόνο σε δευτερόλεπτα θα πρέπει να διαιρέσετε με τη σταθερά **CLOCKS_PER_SEC**. Στο παραπάνω παράδειγμα, θα μπορούσατε να γράψετε:

```
printf("<A> needed time = %d sec", (t2 - t1) / CLOCKS_PER_SEC);
```

Προσοχή (1):

αν μετράτε μικρούς χρόνους (εκατοστα ή δέκατα του δευτερολέπτου), τότε δεν πρέπει να κάνετε ακέραιες διαιρέσεις, γιατί πχ. αν **CLOCKS_PER_SEC** = 100 και t2 - t1 = 20, το παραπάνω θα σας πει ότι χρειάστηκαν 0 δευτερόλεπτα! Ο σωστός τρόπος, επομένως, είναι:

```
printf("<A> needed CPU time = %lf",
    ((double) (t2-t1)) / ((double) CLOCKS_PER_SEC));
```

το οποίο θα σας τυπώσει το 0.2 sec.

Προσοχή (2):

επειδή η **clock()** επιστρέφει ακέραιο, αν το πρόγραμμά σας χρειάζεται πολλή ώρα να τρέξει υπάρχει κίνδυνος να μηδενιστεί ο χρονομετρητής και να λάβετε λάθος μετρήσεις. Π.χ. στο solaris αναφέρεται ότι ο χρονομετρητής μηδενίζεται και μετρά πάλι από την αρχή κάθε 36 λεπτά καθαρού χρόνου εκτέλεσης!

ΟΛΟΚΛΗΡΩΜΕΝΟ ΠΑΡΑΔΕΙΓΜΑ - Χρονομέτρηση ενός αθροίσματος με τη clock()

```
#include <stdio.h> /* Διότι θα χρησιμοποιήσουμε την printf() */
#include <time.h> /* Διότι θα χρησιμοποιήσουμε την clock() */

main()
{
 double t1, t2; /* Τα θέλουμε double, όπως είδαμε παραπάνω */
 int i, sum = 0;

 t1 = (double) clock(); /* Διότι η clock() επιστρέφει clock_t (συνήθως int ή long) */

  for (i = 0; i < 100000000; i++)
    sum += i; /* Απλά προσθέτουμε μέχρι το 100.000.000 */

 t2 = (double) clock();

 printf("1000000 αριθμούς τους πρόσθεσα σε: %lf sec (CPU time).\n",
      (t2 - t1) / CLOCKS_PER_SEC );
}
```

3. Στην πράξη (II) - η times() για μέτρηση CPU time και πραγματικού χρόνου

Ένας γενικότερος τρόπος (από την **clock()**) να χρονομετρήσετε τα προγράμματά σας στο Unix, είναι να κάνετε χρήση της συνάρτησης **times()** από την οποία μπορείτε να υπολογίσετε και το CPU time αλλά και τον πραγματικό χρόνο. Θα πρέπει να συμβουλευτείτε τα manual pages για να μάθετε περισσότερα, αλλά εδώ σας δίνονται σχεδόν όλα όσα θα χρειαστείτε. Για να χρησιμοποιήσετε την **times()** θα πρέπει να κάνετε **#include <sys/times.h>** στο πρόγραμμά σας. Η **times()** επιστρέφει τα clock ticks που έχουν περάσει από κάποια χρονική στιγμή (συνήθως από τότε που ξεκίνησε το σύστημα). Αν επομένως βάλετε την **times()** αμέσως πριν και αμέσως μετά το σημείο **A** στο πρόγραμμά σας:

```
t1 = times(&tb1);
<κώδικας A>
t2 = times(&tb2);
```

η διαφορά των τιμών που θα πάρετε (t2 - t1) σας δίνει ακριβώς πόσα clock ticks χρειάστηκε (σε πραγματικό χρόνο) το **A** για να εκτελεστεί. Από την **times()** μπορείτε να βρείτε και άλλα πράγματα, όπως π.χ. τον χρόνο που σπατάλησε στην CPU το κομμάτι **A** κλπ., αρκεί να προσπελάσετε τα πεδία του tb. Τα tb1/2 είναι μία παράμετρος που δίνεται στην **times()** για να πάρουμε αυτές τις έξτρα πληροφορίες και είναι τύπου **struct tms**:

```
struct tms { clock_t tms_utime, tms_stime, tms_cutime, tms_cstime; };
```

Το άθροισμα tms_utime + tms_stime δίνει το CPU time, και άρα η διαφορά

```
cpu_time = (tb2.tms_utime + tb2.tms_stime) - (tb1.tms_utime + tb1.tms_stime);
```

μας δίνει το CPU time για το κομμάτι **A**.

Για να βρείτε, όμως, τον χρόνο σε δευτερόλεπτα, θα πρέπει να γνωρίζεται κάθε clock tick πόσα δευτερόλεπτα είναι. Επειδή αυτό διαφέρει από σύστημα σε σύστημα, θα χρειαστεί να το βρείτε κάνοντας χρήση της κλήσης **sysconf(_SC_CLK_TCK)**, η οποία σας επιστρέφει πόσα clock ticks αντιστοιχούν σε 1 sec, *κάτι που δεν είναι ίδιο με το **CLOCKS_PER_TICK** της **clock()***! Η συνάρτηση **sysconf()** απαιτεί να κάνετε **#include <unistd.h>.** Εάν υποθέσουμε ότι έχουμε κάνει:

```
ticspersec = sysconf(_SC_CLK_TCK);
```

τότε, π.χ. στο παραπάνω παράδειγμα, θα μπορούσατε να γράψετε:

```
printf("<A> needed time = %d sec (REAL time)", (t2 - t1) / ticspersec);
```

Προσοχή, όμως, όπως και στην clock():

αν μετράτε μικρούς χρόνους (εκατοστα ή δέκατα του δευτερολέπτου), τότε δεν πρέπει να κάνετε ακέραιες διαιρέσεις, γιατί πχ. αν ticspersec = 100 και t2 - t1 = 20, το παραπάνω θα σας πει ότι χρειάστηκαν 0 δευτερόλεπτα! Ο σωστός τρόπος, επομένως, είναι:

```
printf("<A> needed time = %lf",
    ((double) (t2-t1)) / ((double) ticspersec));
```

το οποίο θα σας τυπώσει το 0.2 sec.

ΟΛΟΚΛΗΡΩΜΕΝΟ ΠΑΡΑΔΕΙΓΜΑ - Χρονομέτρηση ενός αθροίσματος με την times()

```
#include <stdio.h> /* Διότι θα χρησιμοποιήσουμε την printf() */
#include <sys/times.h> /* Διότι θα χρησιμοποιήσουμε την times() */
#include <unistd.h> /* Διότι θα χρησιμοποιήσουμε την sysconf() */

main()
{
 double t1, t2, cpu_time; /* Τα θέλουμε double, όπως είδαμε παραπάνω */
 struct tms tb1, tb2;     /* Το χρειάζεται η times() */
 double ticspersec;       /* Το κάνουμε double, όπως είδαμε παραπάνω */
 int    i, sum = 0;

 ticspersec = (double) sysconf(_SC_CLK_TCK);  /* Διότι η sysconf() επιστρέφει int */

 t1 = (double) times(&tb1);          /* Διότι η times() επιστρέφει (long) int */

   for (i = 0; i < 100000000; i++)
     sum += i; /* Απλά προσθέτουμε μέχρι το 100.000.000 */
```

```
  t2 = (double) times(&tb2);

  cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                       (tb1.tms_utime + tb1.tms_stime));
  printf("Χρειάστηκα %lf sec (REAL time) αν και χρησιμοποίησα μόνο %lf sec (CPU time).\n",
         (t2 - t1) / ticspersec, cpu_time / ticspersec);
}
```

Προσέξτε μόνο ότι έχουμε υποθέσει ότι η διεργασία μας δεν έχει δημιουργήσει άλλες διεργασίες - παιδιά. Αν αυτό δεν ισχύει και ενδιαφερόμαστε να χρονομετρήσουμε και το CPU time των παιδιών, τότε πρέπει να χρησιμοποιήσουμε και τα άλλα δύο πεδία του **struct tms**. Διαβάστε προσεκτικά τις προϋποθέσεις στα manual pages.

ΡΥΘΜΙΣΕΙΣ ΓΙΑ ΤΗΝ ΠΡΩΤΗ ΧΡΗΣΗ:

Το LAM/MPI απαιτεί τον ορισμό μίας environmental μεταβλητής καθώς και την τροποποίηση του PATH σας. Όσοι χρησιμοποιούν csh ή tcsh θα πρέπει να τοποθετήσουν τις παρακάτω εντολές στο αρχείο .cshrc ή .tcshrc:

```
# Προσθήκη του /opt/lam/bin στο PATH και ορισμός του LAMHOME
setenv PATH ${PATH}:/opt/lam/bin
setenv LAMHOME /opt/lam
setenv MANPATH ${MANPATH}:/opt/lam/man
```

ενώ όσοι χρησιμοποιούν sh ή bash, θα πρέπει να προσθέσουν τα παρακάτω στο .profile ή .bashrc τους:

```
# Προσθήκη του /opt/lam/bin στο PATH και ορισμός του LAMHOME
PATH=${PATH}:/opt/lam/bin
export PATH
LAMHOME=/opt/lam
export LAMHOME
MANPATH=${MANPATH}:/opt/lam/man
export MANPATH
```

Προσοχή:

- Για να λειτουργήσει το LAM/MPI απαγορεύεται να έχετε στα παραπάνω αρχεία (.cshrc, .bashrc κλπ.) εντολές που τυπώνουν κάτι στο standard output ή στο standard error. Ελέγξτε τα και σιγουρευτείτε ότι δεν υπάρχουν τέτοιες εντολές (όπως π.χ. echo, stty κλπ.).
- Η εγκατάσταση του πακέτου κάνει χρήση του **ssh**, και όχι του **rsh**. Όμως, *θα πρέπει να μην σας ζητάει password το ssh*. Απαιτείται ειδική διαδικασία ώστε να γίνει αυτό. Για να δείτε αν πρέπει να κάνετε αυτή τη διαδικασία, δοκιμάστε το εξής: ας πούμε ότι δουλεύετε στο μηχάνημα eros. Εκτελέσετε ssh προς ένα άλλο μηχάνημα, π.χ.

  ```
  % ssh kerveros ls
  ```

  Αν δεν σας ζητηθεί password τελειώσατε, αλλιώς, θα πρέπει να κάνετε τη διασικασία ρύθμισης του ssh*.

ΔΗΜΙΟΥΡΓΙΑ ΕΙΚΟΝΙΚΗΣ ΠΑΡΑΛΛΗΛΗΣ ΜΗΧΑΝΗΣ:

Θα πρέπει να δημιουργήσετε ένα αρχείο με τα μηχανήματα που θέλετε να "συμμετάσχουν" στον εικονικό παράλληλο υπολογιστή σας (ένα όνομα σε κάθε γραμμή του αρχείου). Για παράδειγμα, θα μπορούσατε να έχετε το παρακάτω αρχείο, που ας πούμε ότι ονομάζεται mymachines, και το οποίο δημιουργεί μία "παράλληλη μηχανή" με τρεις "επεξεργαστές":

```
# Αρχείο 'mymachines' -- σχόλια επιτρέπονται μετά το '#'
# Θέλω τα τρία παρακάτω μηχανήματα:
kerveros
skotos
eros
```

Για να ελέγξετε ότι τα μηχανήματα αυτά όντως μπορούν να συμμετάσχουν, εκτελέστε:

```
% recon -v mymachines
```

Αν όλα πάνε καλά, μπορείτε να προχωρήσετε στην αρχικοποίηση του LAM/MPI. Αν όχι, συμβουλευτείτε τα παρακάτω:

Λίστα πιθανών προβλημάτων:

1. Μέσα στο αρχείο αυτό θα πρέπει οπωσδήποτε να περιλαμβάνεται και το όνομα του μηχανήματος από το οποίο εκτελείται το recon και από το οποίο θα αρχικοποιηθεί το LAM/MPI.
2. Δεν επιτρέπεται ένα μηχάνημα να εμφανιστεί παραπάνω από μία φορά στο αρχείο αυτό.
3. Ελέγξτε ότι το ssh λειτουργεί σωστά και ότι δεν σας ζητάει password (βλ. παραπάνω).
4. Ελέγξτε ότι στα αρχεία αρχικοποίησης του SHELL σας δεν υπάρχουν εντολές που τυπώνουν κάτι (π.χ. echo, stty -- βλ. παραπάνω).

ΑΡΧΙΚΟΠΟΙΗΣΗ ΚΑΙ ΤΕΡΜΑΤΙΣΜΟΣ ΤΟΥ LAM/MPI:

Πριν μπορέσετε να εκτελέσετε τα προγράμματά σας, θα πρέπει να *αρχικοποιήσετε* το LAM/MPI, ως εξής:

```
% lamboot -v mymachines
```

όπου το αρχείο mymachines περιέχει τα ονόματα των μηχανών που θα απαρτίσουν τον εικονικό παράλληλο υπολογιστή (βλ. παραπάνω).

Στο τέλος, θα πρέπει να *τερματίσετε* το LAM/MPI, ως εξής:

```
% wipe -v mymachines
```

Παρατηρήσεις:

1. **ΜΗΝ ΞΕΧΝΑΤΕ ΤΟΝ ΤΕΡΜΑΤΙΣΜΟ!!**
2. Αν δεν προσδιορίσετε το αρχείο mymachines κατά την αρχικοποίηση, θα περιληφθεί *μόνο η τρέχουσα μηχανή* (localhost) στον εικονικό παράλληλο υπολογιστής σας.
3. Μετά την αρχικοποίηση μπορείτε να εκτελέσετε όσα προγράμματα επιθυμείτε (δηλαδή απαιτείται μόνο μία αρχικοποίηση) μέχρι να τερματίσετε το LAM/MPI.
4. Εκτελέσετε wipe -v (χωρίς το αρχείο του εικονικού υπολογιστή) για να τερματίσετε το LAM/MPI στην τρέχουσα μηχανή.

ΜΕΤΑΓΛΩΤΙΣΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ (COMPILING):

Σε κάθε πρόγραμμα C θα πρέπει να γίνεται:

```
#include "mpi.h"
```

Μεταγλώτισση γίνεται με την εντολή mpicc, η οποία αναλαμβάνει να περιλάβει όλα τα απαραίτητα αρχεία και βιβλιοθήκες:

```
% mpicc <αρχείο C>
```

Σημείωση: δεν είναι απαραίτητη η αρχικοποίηση του MPI για να μεταγλωτίσει κανείς προγράμματα με το mpicc. Η αρχικοποίηση όμως είναι απαραίτητη για την εκτέλεσή τους.
Εδώ μπορείτε να βρείτε ένα δοκιμαστικό αρχείο, για τον υπολογισμό του 'π' (=3.14...)

ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ:

Εκτελέστε:

```
% mpirun -np N <εκτελέσιμο αρχείο>
```

όπου N είναι ο αριθμός των διεργασιών που θέλετε να δημιουργηθούν. Οι διεργασίες μοιράζονται και εκτελούνται στους κόμβους της εικονικής παράλληλης μηχανής που προσδιορίστηκε κατά την αρχικοποίηση του LAM/MPI. Αν το N είναι μεγαλύτερο από τον αριθμό των μηχανημάτων, η διαμοίραση των εργασιών γίνεται κυκλικά.

*** Ρύθμιση του SSH ώστε να μη ζητά password**
(Αφορά την έκδοση 2 του ssh)

1. Εκτελέστε:

   ```
   % ssh-keygen -t rsa
   ```

   Δώστε τρία enter (δηλαδή να δημιουργήσει το default αρχείο, και να χρησιμοποιήσει κενό pass phrase).

2. Θα δημιουργηθεί ένα directory .ssh στο λογαρισασμό σας Μπείτε στο .ssh και κάντε ls -l. Θα πρέπει να δείτε τα παρακάτω:
3. ```
   -rw-------  id_rsa
   -rw-r--r--  id_rsa.pub
   ```

   Αν το αρχείο id_rsa.pub δεν έχει αυτά τα permissions, θα πρέπει να εκτελέσετε:

   ```
   % chmod 0644 id_rsa.pub
   ```

4. Αντιγράψτε το αρχείο id_rsa.pub ως εξής:

   ```
   % cp id_rsa.pub authorized_keys
   ```

5. Από εδώ και στο εξής, δεν θα πρέπει να σας ζητά password το ssh. Απλά, αν επιθυμείττε να χρησιμοποιήσετε τα μηχανήματα π.χ. Α και Β με το LAM/MPI, εκτελέστε μία φορά ssh προς καθένα από αυτά για να τα "μάθει":
6. ```
   % ssh <μηχάνημα A> ls
   % ssh <μηχάνημα B> ls
   ```

Ίσως το ssh σας ρωτήσει αν όντως επιθυμείτε τη σύνδεση (απαντήστε "yes") και θα προσθέσει τα μηχανήματα Α και Β στη λίστα των "γνωστών" του μηχανημάτων. Την επόμενη φορά που θα κάνετε ssh σε αυτά τα μηχανήματα, δεν θα σας ζητηθεί τίποτε.

Αρχικές ρυθμίσεις

Αρχικά θα πρέπει να ορίσετε την μεταβλητή περιβάλλοντος CLASSPATH. Ο καλύτερος τρόπος για να το κάνετε αυτό είναι να την δηλώσετε στο αρχείο .tcshrc, .bashrc ή .profile, ανάλογα με το πιο shell χρησιμοποιείτε.

- Αν χρησιμοποιείτε το **bash**, τότε είτε στο αρχείο ~/.bashrc είτε στο αρχείο ~/.profile προσθέστε στο τέλος τις εξής εντολές:
- CLASSPATH=/usr/local/mic1/classes.zip
- export CLASSPATH

- Αν χρησιμοποιείτε το **tcsh**, τότε στο αρχείο ~/.tcshrc προσθέστε στο τέλος την εξής εντολή:
- setenv CLASSPATH /usr/local/mic1/classes.zip

(Αν δεν γνωρίζεται το shell σας, μπορείτε να το βρείτε εκτελώντας: echo $SHELL).

Αντιγραφή βασικών αρχείων

Θα πρέπει να δημιουργήσετε κάποιο directory στο οποίο θα δουλεύετε, π.χ. ~/ARCHITECTURE. Θα πρέπει να πάτε εκεί και αντιγράψετε τα εξής τρία αρχεία:

```
cd ARCHITECTURE
cp /usr/local/mic1/ijvm.conf .
cp /usr/local/mic1/mic1ijvm.mic1 .
cp /usr/local/mic1/mic1ijvm.mal .
```

Δουλεύοντας με τον εξομοιωτή

- Για να μεταγλωτίσετε ένα πρόγραμμα prog.jas γραμμένο σε συμβολική γλώσσα μηχανής της IJVM, θα πρέπει να χρησιμοποιήσετε την εντολή:

  ```
  java ijvmasm prog.jas
  ```

  Θα δημιουργηθεί έτσι ένα μεταφρασμένο αρχείο prog.ijvm το οποίο είναι κατάλληλο για φόρτωμα στον εξομοιωτή.

- Για να ξεκινήσετε τον εξομοιωτή, θα πρέπει να εκτελέσετε:

  ```
  java mic1sim mic1ijvm.mic1
  ```

  Από την επιλογή File/Load Macroprogram μπορείτε να φορτώσετε ένα μεταφρασμένο πρόγραμμα (π.χ. το prog.ijvm) και να το εκτελέσετε.

Μικροπρογραμματισμός

Προκειμένου να κάνετε τον εξομοιωτή να δουλεύει με ένα νέο μικροπρόγραμμα, απαιτούνται τα παρακάτω:

- Αν δεν το έχετε ήδη κάνει, αντιγράψτε τα παρακάτω αρχεία στο directory που δουλεύετε:

- cd ARCHITECTURE
- cp /usr/local/mic1/ijvm.conf .
- cp /usr/local/mic1/mic1ijvm.mal .
- mv mic1ijvm.mal mine.mal

όπου μετονομάσαμε το mic1ijvm.mal σε mine.mal για ευκολία. Σιγουρευτείτε ότι μπορείτε να τροποποιήσετε αυτά τα αρχεία. Εν ανάγκη εκτελέστε:

```
chmod +w ijvm.conf
chmod +w mine.mal
```

- Ότι νέες εντολές γλώσσας μηχανής (επιπρόσθετα από αυτές της IJVM) έχετε να προσθέσετε θα πρέπει να τις δηλώσετε στο αρχείο ijvm.conf. Για κάθε εντολή πρέπει να δίνεται και ο κωδικός της (εσείς τον αποφασίζετε).
- Τροποποιείστε το μικροπρόγραμμα (mine.mal). Όπως θα δείτε, θα πρέπει πρώτα να δηλώσετε τα αρχικά labels για τις μικροεντολές που υλοποιούν τις νέες εντολές γλώσσας μηχανής. Στη συνέχεια, μπορείτε να προσθέσετε τον νέο μικροκώδικά σας σε όποιο σημείο του αρχείου θέλετε.
- Το νέο σας μικροπρόγραμμα θα πρέπει να το *μικρο-μεταφράσετε*! Εκτελέστε:
- java mic1asm mine.mal mine.mic1

και θα δημιουργηθεί ένα νέο αρχείο μικροπρογράμματος mine.mic1 το οποίο μπορεί να χρησιμοποιήσει ο εξομοιωτής.

- Τέλος, για να τρέξετε τον εξομοιωτή με το νέο μικροπρόγραμμα εκτελέστε:
- java mic1sim mine.mic1

Πλέον μπορείτε να γράψετε προγράμματα σε γλώσσα μηχανής που χρησιμοποιούν τις νέες εντολές και να τα εκτελέσετε!

```
/* GRAPHFILE.C
 * ==========
 * Functions to open, read and close a file containing a graph description.
 * 1. First use graph_openfile() to open the file and retrieve
 *    the number of vertices & edges.
 * 2. Then loop with graph_nextvertex() to get each vertex's degree
 *    and neighbors
 * 3. Stop the loop when graph_nextvertex() returns EOF.
 * 4.  Finally, DO NOT FORGET to do graph_closfile().
 *
 * The functions may return 1 if some error was encountered.
 *
 * V. V. Dimakopoulos, CS, UoI, 1999 - 2000
 */


#include <ctype.h>
```

```
static FILE *infp = (FILE *) NULL;
static char _linestr[1000], *_curpos;
static int  _fline;


graph_openfile(char *name, int *n, int *m)
{
 *n = *m = _fline = 0;
 if ((infp = fopen(name, "r")) == NULL)
 {
  fprintf(stderr, "Δεν μπορώ να διαβάσω το `%s' ..\n", name);
  return (1);
 }
 return ( _get_nm(n, m) );
}


graph_closefile()
{
 fclose(infp);
 infp = NULL;
 _fline = 0;
}


graph_nextvertex(int *v, int *d, int neighbors[])
{
 int i;

 *v = *d = 0;
 if (_getline() == EOF) return (EOF);
 if ((*v = _getnumber()) != -1)
  if ((*d = _getnumber()) != -1)
  {
   for (i = 0; i < *d; i++)
    if ((neighbors[i] = _getnumber()) == -1)
        {
         fprintf(stderr, "Είτε λάθος ακέραιος είτε λιγότεροι γείτονες απότι πρέπει ..\n");
         return (1);
        };
   return (0);
  }
 fprintf(stderr, "Κάθε γραμμή του αρχείου πρέπει να συντάσεται ως εξής:\n");
 fprintf(stderr, "κορυφή βαθμός γείτονας γείτονας ... γείτονας\n");
 return (1);
}


_get_nm(int *n, int *m)
{
 *n = *m = 0;
 if (_getline() == EOF)
 {
  fprintf(stderr, " Το αρχείο είναι άδειο ..\n");
  return (1);
 }
 if ((*n = _getnumber()) != -1)
  if ((*m = _getnumber()) != -1)
   return (0);
 fprintf(stderr, "Η γραμμή %d έπρεπε να περιέχει τα n και m ..\n", _fline);
```

```
 return (1);
}


_getline()
{
 char *s;

 for (;;)
 {
  if (fgets(_linestr, 999, infp) == NULL) return (EOF);
  _fline++;
  /* Skip spaces & check for comments */
  for (s = _linestr; isspace(*s); s++)
    ;
  if (*s)   /* I.e. not end of string (empty line) */
    if (*s != '#')  /* I.e. not a comment line */
    {
      _curpos = s;
      return (0);   /* Line is ok */
    }
 }  /* If *s = 0 or *s = #, skip line and read next one */
}


_getnumber()
{
 int  x;
 char *s;

 /* Skip spaces */
 for (; isspace(*_curpos); _curpos++)
   ;
 if (*_curpos) /* not end of string (line) */
   if (isdigit(*_curpos))
   {
     s = _curpos;
     for (; isdigit(*_curpos); _curpos++)   /* Keep digits only */
       ;
     if (!isalpha(*_curpos))
     {
       sscanf(s, "%d", &x);
       return (x);
     }
   }
 fprintf(stderr, "Περίμενα ένα αριθμό στη γραμμή %d του αρχείου ..\n", _fline);
 return (-1);
}

* TESTGR
 * ======
 * Ένα απλό πρόγραμμα που δείνχει πως να χρησιμοποιήσετε το
 * 'graphfile.c'. Απλά, διαβάζει ένα γράφο από ένα αρχείο
 * και τυπώνει τις κορυφές με τις γειτονιές τους.
 *
 * Ο ΚΩΔΙΚΑΣ ΕΙΝΑΙ ΕΛΑΧΙΣΤΟΣ, ΑΠΛΑ ΕΧΕΙ ΠΟΛΛΑ ΣΧΟΛΙΑ!!!
 *
 * Β.Β. Δημακόπουλος (Ε-19, ¶νοιξη 2000)
 */
```

```
#include <stdio.h>
#include "graphfile.c"


main()
{
 int neighbors[50];
 int n, m, v, d, sum = 0, i, result;
 char fname[20];

 printf("Δώστε το αρχείο που έχει το γράφο: "); scanf("%s", fname);

 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Η graph_openfile() επιστρέφει 1 αν κάτι δεν πάει καλά              *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 if (graph_openfile(fname, &n, &m) == 1)
   exit (1);

 printf("Ο γράφος έχει %d κορυφές και %d ακμές\n", n, m);

 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Η graph_nextvertex() επιστρέφει 0, κανονικά.                       *
  * Αν επιστρέψει κάτι άλλο θα είναι είτε EOF (στο τέλος του αρχείου)   *
  * είτε κάτι άλλο (αν συμβεί κάποιο πρόβλημα).                        *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 while ( (result = graph_nextvertex(&v, &d, neighbors)) == 0)
 {
  printf("Η κορυφή %d έχει βαθμό %d και γείτονες: ", v, d);
  sum += d;
  for (i = 0; i < d; i++)
    printf("%d, ", neighbors[i]);
  printf("\n");
 }

 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Πρέπει πάντα να κλείνουμε το αρχείο που ανοίξαμε                   *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 graph_closefile();

 if (result != EOF) exit (1);    /* Error in file */

 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Για να σιγουρευτούμε ότι το αρχείο περιείχε όντως ένα σωστό γράφο,  *
  * θα πρέπει να κάνουμε κάποιου απλούς ελέγχους, όπως π.χ.             *
  * (1) ο αριθμός των ακμών θα πρέπει να είναι ίσος με το άθροισμα      *
  *     των βαθμών, δια δύο                                            *
  * (2) αν η ακμή (α,β) δόθηκε, θα πρέπει να έχει δοθεί και η (β,α)     *
  *     (ώστε ο πινακας γειτνίασης να είναι συμμετρικός)               *
  * (3) κλπ κλπ.                                                       *
  * Παρακάτω ελέγχω το (1):                                            *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 if (sum != 2*m)
 {
  printf("Κάπου υπάρχει λάθος: m = %d ενώ Σd(v) = %d\n", m, sum);
  exit (1);
 }
}

// Simple program to add numbers from 1 to 10
//
```

```
// int i, sum;
//
// sum = 0;
// for (i = 0; i <= 10; i = i+1)
//   sum = sum+i;
//
// V.V. Dimakopoulos

.main
  .var
    i
    sum
  .end-var

                BIPUSH 0        // sum = 0
                ISTORE sum
                BIPUSH 0        // i = 0
                ISTORE i

  loop:
                BIPUSH 11       // if i == 11, goto end
                ILOAD  i
                IF_ICMPEQ end

                ILOAD  i        // sum = sum + i
                ILOAD  sum
                IADD
                ISTORE sum

                BIPUSH 1        // i = i + 1
                ILOAD  i
        IADD
                ISTORE i

                GOTO   loop

  end:
        ILOAD  sum
                OUT             // Should be 55, which is ASCII '7'
        HALT
.end-main

/* pi.c
 * ----
 *   Serial pi calculation.
 *
 * VVD, 2000 (c)
 */
#include <stdio.h>

#define N     10000000      /* Intervals */


main()
{
  int    i;
  double W = 1.0 / ((double) N), W2, pi = 0.0;

  W2 = W*W;   /* Save some calculations */
  W = 4*W;
```

```
  for (i = 0; i < N; i ++)
    pi += W / (1.0 + (0.5 + (double) i)*(0.5 + (double) i)*W2);
  printf("pi = %.10lf\n", pi);
}
```

```
function [pmf, x] = kernelestim(data, ktype, bins, bandwidth);
%
% Estimates the probability mass function (pmf)
% with the kernel estimation method.
% The following kernel types are allowed:
%
%   - ktype='hist'    : histogram (bandwidth isn't needed)
%   - ktype='rect'    : rectangular kernel
%                       (with the histogram as a special case)
%   - ktype='gauss'   : Gauss kernel
%
% pmf      : estimated pmf
% x        : bin positions
% data     : data vector
% ktype    : kernel type
% bins     : number of bins
% bandwidth: "bandwidth" of the estimator, i.e. width of the kernel.
%            (Gauss kernel uses all data, hence the 'bandwidth'
%            argument is ignored.)
%

[N M] = size(data);
if N == 1
  data = data';
  N = M;
elseif N > 1 & M > 1
  disp('O pinakas twn dedomenwn eisodou xreiazetai anastrofi!');
  return;
end;

minx = min(data);
maxx = max(data);
x = minx: (maxx - minx) / bins: maxx;

for k = 1: length(x)
  XX = x(k)*ones(N,1);
  arg = XX - data;

  switch ktype(1)
  case 'h'
    if k==1
       xmin=minx-1;
    else
       xmin=x(k-1);
    end
    mask = find(data>=xmin & data<x(k));
    pmf(k) = length(mask);
  case 'r'
    mask = find(abs(arg) < bandwidth/2);
    pmf(k) = length(mask);
  case 'g'
    pmf(k) = sum(normpdf(arg/bandwidth,0,1));
  end;
```

```
end;

switch ktype(1)
  case 'h'
     pmf = pmf / (N*abs((maxx-minx)/bins));
   otherwise
     pmf = pmf / (N * bandwidth);
end
#include "main.h"

main()
{
 int i,x,y,x1,y1;
 point pos;

 SRGP_begin(APL_NAME,WIN_X_SIZE,WIN_Y_SIZE,COLORS,ENABLE_TRACE);
 Initialize();
 CreateMenuBar();
 CreateColorPalette();
 SRGP_refresh();

 SRGP_setInputMode(LOCATOR,EVENT);
 while(1)
 {
  SRGP_waitEvent(-1);
  SRGP_getLocator(&LocMeasure);
  if(LocMeasure.button_chord[0]==1)
   {
    x=LocMeasure.position.x;
    y=LocMeasure.position.y;

    if(x>=9&&x<=89&&y>=8&&y<=43&&DialogFlag==False)
     {
      if(MenuButton.form==Up)
       {
        MenuFlag=True;
        MenuButton.form=Down;
        SaveCanvas();
        CreateButton(MenuButton);
        CreatePopUpMenu();
       }
      else
       {
        MenuFlag=False;
        MenuButton.form=Up;
        CreateButton(MenuButton);
        ClearPopUpMenu();
       }
     }
    else if(MenuFlag==True)
     {
      if(x<1||x>150||y<51||y>200)
       {
        SRGP_beep();
       }
      else if(y>150)
       {
        MenuFlag=False;
        MenuButton.form=Up;
        CreateButton(MenuButton);
```

```
    ClearPopUpMenu();
    DialogFlag=True;
    OpenFlag=True;
    CreateDialog("Φόρτωση Αρχείου");
   }
  else if(y>100)
   {
    MenuFlag=False;
    MenuButton.form=Up;
    CreateButton(MenuButton);
    ClearPopUpMenu();
    DialogFlag=True;
    SaveFlag=True;
    CreateDialog("Αποθήκευση Αρχείου");
   }
  else
   exit(0);
 }
else if(DialogFlag==True)
 {
 pos.x=315;
 pos.y=310;
 SRGP_setKeyboardEchoOrigin(pos);
 SRGP_setInputMode(KEYBOARD,SAMPLE);
 SRGP_sampleKeyboard(FileName,30);

 if(x<300||x>500||y<250||y>400)
  {
   SRGP_beep();
  }
 else if(x>=330&&x<=390&&y>=260&&y<=285)
  {
   DialogFlag=False;
   FlushButton(OkBtn);
   SRGP_setInputMode(KEYBOARD,INACTIVE);
   SRGP_setKeyboardMeasure("");
   ClearDialog();
   if(OpenFlag==True)
    {
     OpenFile(FileName);
     OpenFlag=False;
    }
   else
    {
     SaveFile(FileName);
     SaveFlag=False;
    }
  }
 else if(x>=410&&x<=470&&y>=260&&y<=285)
  {
   DialogFlag=False;
   FlushButton(CancelBtn);
   SRGP_setInputMode(KEYBOARD,INACTIVE);
   SRGP_setKeyboardMeasure("");
   ClearDialog();
   OpenFlag=False;
   SaveFlag=False;
  }
 }
else if(x>=200&&x<=635&&y>=8&&y<=43)
```

```
    {
     x=(x-199)/55;

     if(x==0)
      {
       FlushButton(button[x]);
       ClearPanel();
      }
     else if(x==1)
      {
       FlushButton(button[x]);
       SRGP_refresh();
      }
     else if((x-2)!=DesignFlag)
      {
       if(DesignFlag!=NONE)
        {
         button[DesignFlag+2].form=Up;
         CreateButton(button[DesignFlag+2]);
        }
       button[x].form=Down;
       CreateButton(button[x]);

       DesignFlag=x-2;
      }
    }
   else if(DesignFlag!=NONE && x<WIN_X_SIZE-50 && y>50)
    {
     if(DesignFlag==COPY)
      CopyRegion(x,y);
     else if(DesignFlag==LINE)
      CreateLine(x,y);
     else if(DesignFlag==CIRCLE)
      CreateCircleArc(x,y);
     else if(DesignFlag==ELLIPSE)
      CreateEllipseArc(x,y);
     else if(DesignFlag==PARABOLE)
      CreateParaboleArc(x,y);
     else if(DesignFlag==YPERBOLE)
      CreateYperboleArc(x,y);
    }
   else if(x>=(WIN_X_SIZE-44)&&x<=(WIN_X_SIZE-8)&&y>=55&&y<=645)
    {
     y=(y-54)/37;

     if(color_button[y].form==Up)
      {
       color_button[y].form=Down;
       color_button[LastColor].form=Up;
       CreateButton(color_button[y]);
       CreateButton(color_button[LastColor]);
       LastColor=y;
       SRGP_setColor(LastColor);
      }
    }
   }
  }

  SRGP_end();
}
```

```
void Initialize()
{
 int i;

 LoadCommonColors();
 SRGP_loadFont(1,"lucidasansgreek");
 SRGP_setFont(1);

 DesignFlag=NONE;

 MenuButton.form=Up;
 MenuButton.x=9;
 MenuButton.y=8;
 MenuButton.dx=80;
 MenuButton.dy=35;
 MenuButton.color=GOLD;
 MenuButton.tag=0;

 OkBtn.form=Up;
 OkBtn.x=330;
 OkBtn.y=260;
 OkBtn.dx=60;
 OkBtn.dy=25;
 OkBtn.color=GOLD;
 OkBtn.tag=9;

 CancelBtn.form=Up;
 CancelBtn.x=410;
 CancelBtn.y=260;
 CancelBtn.dx=60;
 CancelBtn.dy=25;
 CancelBtn.color=GOLD;
 CancelBtn.tag=10;

 for(i=0;i<BUTTONS;i++)
  {
  button[i].form=Up;
  button[i].x=200+i*55;
  button[i].y=8;
  button[i].dx=50;
  button[i].dy=35;
  button[i].color=GRAY;
  button[i].tag=i+1;
  }

 for(i=0;i<COLOR_BUTTONS;i++)
  {
  if(i==1)
   {
    color_button[i].form=Down;
    LastColor=1;
   }
  else
   color_button[i].form=Up;
  color_button[i].x=WIN_X_SIZE-44;
  color_button[i].y=55+i*37;
  color_button[i].dx=36;
  color_button[i].dy=35;
  color_button[i].color=i;
```

```
  color_button[i].tag=-1;
 }

 SRGP_setColor(LastColor);
}

void LoadCommonColors()
{
 SRGP_loadCommonColor(2,"SkyBlue");
 SRGP_loadCommonColor(3,"LightBlue");
 SRGP_loadCommonColor(4,"Blue");
 SRGP_loadCommonColor(5,"Cyan");
 SRGP_loadCommonColor(6,"Green");
 SRGP_loadCommonColor(7,"SeaGreen");
 SRGP_loadCommonColor(8,"Yellow");
 SRGP_loadCommonColor(9,"Red");
 SRGP_loadCommonColor(10,"Orange");
 SRGP_loadCommonColor(11,"Maroon");
 SRGP_loadCommonColor(12,"Gray");
 SRGP_loadCommonColor(13,"LightGray");
 SRGP_loadCommonColor(14,"Gold");
 SRGP_loadCommonColor(15,"Coral1");
}

void CreateMenuBar()
{
 FORM form=Up;
 int i;

 SRGP_setColor(SRGP_WHITE);
 SRGP_fillRectangleCoord(0,0,WIN_X_SIZE,50);
 SRGP_setColor(LastColor);
 CreateRect3D(1,1,WIN_X_SIZE-1,50,SKYBLUE,form);

 CreateButton(MenuButton);

 for(i=0;i<BUTTONS;i++)
  CreateButton(button[i]);

 ClearHint();
}

void CreatePopUpMenu()
{
 point pos;

 CreateRect3D(1,51,150,200,GRAY,Up);
 SRGP_setColor(SRGP_BLACK);

 pos.x=30;
 pos.y=170;
 SRGP_text(pos,"Φόρτωση...");

 pos.y=120;
 SRGP_text(pos,"Αποθήκευση ως...");

 pos.y=70;
 SRGP_text(pos,"Εξοδος");

 SRGP_setColor(SRGP_WHITE);
```

```
 SRGP_lineCoord(7,100,144,100);
 SRGP_lineCoord(7,150,144,150);
 SRGP_setColor(LastColor);
}

void ClearPopUpMenu()
{
 ReloadCanvas();
}

void CreateColorPalette()
{
 int i;
 FORM form=Up;

 SRGP_setColor(SRGP_WHITE);
 SRGP_fillRectangleCoord(WIN_X_SIZE-50,51,WIN_X_SIZE,WIN_Y_SIZE);
 SRGP_setColor(LastColor);
 CreateRect3D(WIN_X_SIZE-50,52,WIN_X_SIZE-2,WIN_Y_SIZE-1,LIGHTGRAY,form);

 for(i=0;i<COLOR_BUTTONS;i++)
  {
   CreateButton(color_button[i]);
  }
}

void CreateRect3D(int x1,int y1,int x2,int y2,int color,FORM form)
{
 int x_coords[7];
 int y_coords[7];

/* Δημιουργία εσωτερικού γεμίσματος της 3D επιφάνειας */
 SRGP_setColor(color);
 SRGP_fillRectangleCoord(x1+3,y1+3,x2-3,y2-3);

/* Δημιουργία ημιπεριγράμματος της 3D επιφάνειας */
 if(form==Up)
  SRGP_setColor(SRGP_WHITE);
 else
  SRGP_setColor(SRGP_BLACK);

 x_coords[0]=x1;
 x_coords[1]=x1;
 x_coords[2]=x2;
 x_coords[3]=x2-2;
 x_coords[4]=x1+2;
 x_coords[5]=x1+2;
 x_coords[6]=x1;

 y_coords[0]=y1;
 y_coords[1]=y2;
 y_coords[2]=y2;
 y_coords[3]=y2-2;
 y_coords[4]=y2-2;
 y_coords[5]=y1+2;
 y_coords[6]=y1;

 SRGP_fillPolygonCoord(7,x_coords,y_coords);
```

```
/* Δημιουργία ημιπεριγράμματος της 3D επιφάνειας */
 if(form==Up)
  SRGP_setColor(SRGP_BLACK);
 else
  SRGP_setColor(SRGP_WHITE);

 x_coords[0]=x1;
 x_coords[1]=x2;
 x_coords[2]=x2;
 x_coords[3]=x2-2;
 x_coords[4]=x2-2;
 x_coords[5]=x1+2;
 x_coords[6]=x1;

 y_coords[0]=y1;
 y_coords[1]=y1;
 y_coords[2]=y2;
 y_coords[3]=y2-2;
 y_coords[4]=y1+2;
 y_coords[5]=y1+2;
 y_coords[6]=y1;

 SRGP_fillPolygonCoord(7,x_coords,y_coords);
 SRGP_setColor(LastColor);
}

void CreateButton(Button btn)
{
 point pos,pos1;
 rectangle rect;
 int x1,x2,y1,y2,color;
 FORM form;

 x1=btn.x;
 y1=btn.y;
 x2=btn.x+btn.dx;
 y2=btn.y+btn.dy;
 color=btn.color;
 form=btn.form;

 CreateRect3D(x1,y1,x2,y2,color,form);

 SRGP_setColor(SRGP_BLACK);

 if(btn.tag==0)
  {
   pos.x=x1+20;
   pos.y=y1+15;
   SRGP_text(pos,"Αρχείο");
  }
 else if(btn.tag==1)
  {
   pos.x=x1+20;
   pos.y=y1+15;
   SRGP_text(pos,"K");
  }
 else if(btn.tag==2)
  {
   pos.x=x1+20;
   pos.y=y1+15;
```

```
  SRGP_text(pos,"A");
 }
else if(btn.tag==3)
 {
 SRGP_rectangleCoord(x1+10 ,y1+15,x2-20,y2-10);
 SRGP_rectangleCoord(x1+20,y1+10 ,x2-10 ,y2-15);
 SetHint("Επέλεξε περιοχή για αντιγραφή",
       "και μετά το σημείο προορισμού");
 }
else if(btn.tag==4)
 {
 SRGP_lineCoord(x1+10,y1+10,x2-10,y2-10);
 SetHint("Επέλεξε δύο σημεία για τη",
       "δημιουργία μιας ευθείας");
 }
else if(btn.tag==5)
 {
 pos.x=x1+12;
 pos.y=y1+5;

 pos1.x=x2-12;
 pos1.y=y2-4;

 rect.bottom_left=pos;
 rect.top_right=pos1;

 SRGP_ellipseArc(rect,120,60);
 SetHint("Επέλεξε το κέντρο και",
       "δύο σημεία του κύκλου");
 }
else if(btn.tag==6)
 {
 pos.x=x1+5;
 pos.y=y1+5;

 pos1.x=x2-5;
 pos1.y=y2-5;

 rect.bottom_left=pos;
 rect.top_right=pos1;

 SRGP_ellipseArc(rect,120,60);
 SetHint("Επέλεξε την περιοχή και",
       "δύο σημεία της έλλειψης");
 }
else if(btn.tag==7)
 {
 pos.x=x1+10;
 pos.y=y1+5;

 pos1.x=x2-10;
 pos1.y=y2+20;

 rect.bottom_left=pos;
 rect.top_right=pos1;

 SRGP_ellipseArc(rect,180,0);
 SetHint("Επέλεξε την περιοχή και",
       "δύο σημεία της παραβολής");
 }
```

```
 else if(btn.tag==8)
  {
   pos.x=x1+5;
   pos.y=y1+5;

   pos1.x=x2+20;
   pos1.y=y2+20;

   rect.bottom_left=pos;
   rect.top_right=pos1;

   SRGP_ellipseArc(rect,180,270);
   SetHint("Επέλεξε τεταρτημόριο και",
         "ένα σημείο της υπερβολής");
  }
 else if(btn.tag==9)
  {
   pos.x=x1+7;
   pos.y=y1+8;
   SRGP_text(pos,"Εγκυρο");
  }
 else if(btn.tag==10)
  {
   pos.x=x1+10;
   pos.y=y1+8;
   SRGP_text(pos,"¶κυρο");
  }

 SRGP_setColor(LastColor);
}

void FlushButton(Button btn)
{
 int i;

 btn.form=Down;
 CreateButton(btn);
 SRGP_refresh();
 for(i=0;i<1000000;i++){}
 btn.form=Up;
 CreateButton(btn);
 SRGP_refresh();
 for(i=0;i<1000000;i++){}
}

void OpenFile(char *fname)
{
 FILE *fptr,*tptr;
 char c,command[100]="";
 int x,y,col,percent=0,TotalPoints;

 if((fptr=fopen(fname,"r"))==NULL)
  {
   SetHint("Σφάλμα στην ανάγνωση","του αρχείου εισόδου");
   return;
  }

 ClearPanel();
 ClearHint();
 CreateProgressBar();
```

```
 OldX=663;

 strcat(command,"wc -l ");
 strcat(command,fname);
 strcat(command," > _t");
 system(command);
 tptr=fopen("_t","r");
 fscanf(tptr,"%i",&TotalPoints);
 fclose(tptr);
 system("rm _t");

 while((c=fgetc(fptr))!=EOF)
  {
   ungetc(c,fptr);
   fscanf(fptr,"%i %i %i",&col,&x,&y);
   SRGP_setColor(col);
   SRGP_pointCoord(x,y);
   percent++;

   UpdateProgressBar((percent*100)/TotalPoints);
  }

 SRGP_setColor(LastColor);

 ClearHint();

 fclose(fptr);
}

void SaveFile(char *fname)
{
 FILE *fptr;
 int y,x;
 char col;

 if((fptr=fopen(fname,"w"))==NULL)
  {
   SetHint("Σφάλμα στην εγγραφή","του αρχείου εξόδου");
   return;
  }

 SRGP_getImage(0);

 ClearHint();
 CreateProgressBar();
 OldX=663;

 for(y=WIN_Y_SIZE;y>50;y--)
  {
   for(x=0;x<(WIN_X_SIZE-50);x++)
    {
     col=SRGP_fastGetPix(x,y);
     if(col!=0)
      {
       fprintf(fptr,"%i %i %i\n",col,x,y);
      }
    }
   UpdateProgressBar((WIN_Y_SIZE-y)*100/(WIN_Y_SIZE-50));
  }
```

```
 ClearHint();

 fclose(fptr);
}

void CreateDialog(char *caption)
{
 point pos={345,380};

 SaveCanvas();

 CreateRect3D(300,250,500,400,GRAY,Up);
 CreateRect3D(305,370,495,395,SKYBLUE,Up);
 CreateRect3D(310,300,490,330,SRGP_WHITE,Down);
 CreateButton(OkBtn);
 CreateButton(CancelBtn);
 SRGP_setColor(SRGP_BLACK);
 SRGP_text(pos,caption);
 pos.x=310;
 pos.y=340;
 SRGP_text(pos,"Δώσε το όνομα του αρχείου");
 SRGP_setColor(LastColor);
}

void ClearDialog()
{
 ReloadCanvas();
}

void ClearPanel()
{
 SRGP_setColor(SRGP_WHITE);
 SRGP_fillRectangleCoord(0,51,WIN_X_SIZE-51,WIN_Y_SIZE);
 SRGP_setColor(LastColor);
}

void CopyRegion(int x1,int y1)
{
 int x2=0,y2=0,x3=0,y3=0,i;
 point pos;
 rectangle rect;
 canvasID canvas;

 pos.x=x1;
 pos.y=y1;

 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_RECT);

 do
  {
  SRGP_sampleLocator(&LocMeasure);
  x2=LocMeasure.position.x;
  y2=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);

 SRGP_setLocatorEchoType(CURSOR);
```

```
for(i=0;i<2000000;i++){}

do
 {
 SRGP_sampleLocator(&LocMeasure);
 x3=LocMeasure.position.x;
 y3=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x3>=(WIN_X_SIZE-50) || y3<=50);

canvas=SRGP_inquireActiveCanvas();

pos.x=x3;
pos.y=y3;

if(x1<=x2 && y1<=y2)
 {
 rect.bottom_left.x=x1;
 rect.bottom_left.y=y1;
 rect.top_right.x=x2;
 rect.top_right.y=y2;
 }
else if(x2<x1 && y2<y1)
 {
 rect.bottom_left.x=x2;
 rect.bottom_left.y=y2;
 rect.top_right.x=x1;
 rect.top_right.y=y1;
 }
else if(x1<=x2 && y2<y1)
 {
 rect.bottom_left.x=x1;
 rect.bottom_left.y=y2;
 rect.top_right.x=x2;
 rect.top_right.y=y1;
 }
else if(x2<x1 && y1<=y2)
 {
 rect.bottom_left.x=x2;
 rect.bottom_left.y=y1;
 rect.top_right.x=x1;
 rect.top_right.y=y2;
 }

SRGP_setWriteMode(WRITE_OR);
SRGP_copyPixel(canvas,rect,pos);
SRGP_setWriteMode(WRITE_REPLACE);
CreateColorPalette();

SRGP_setInputMode(LOCATOR,EVENT);
}

void CreateLine(int x1,int y1)
{
 int x2=0,y2=0;
 point pos;

 pos.x=x1;
 pos.y=y1;
```

```
 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_LINE);

 do
  {
   SRGP_sampleLocator(&LocMeasure);
   x2=LocMeasure.position.x;
   y2=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);

 SRGP_lineCoord(x1,y1,x2,y2);

 SRGP_setLocatorEchoType(CURSOR);
 SRGP_setInputMode(LOCATOR,EVENT);
}

void CreateCircleArc(int x1,int y1)
{
 int x2=0,y2=0,x3=0,y3=0,i;
 point pos;
 rectangle rect;
 int radius;
 int StartAngle,EndAngle;

 pos.x=x1;
 pos.y=y1;

 SaveCanvas();

 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_LINE);

 do
  {
   SRGP_sampleLocator(&LocMeasure);
   x2=LocMeasure.position.x;
   y2=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);

 radius=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));

 rect.bottom_left.x=x1-radius;
 rect.bottom_left.y=y1-radius;
 rect.top_right.x=x1+radius;
 rect.top_right.y=y1+radius;

 SRGP_setLineStyle(DOTTED);
 SRGP_lineCoord(x1,y1,x2,y2);
 SRGP_setLineStyle(CONTINUOUS);
 SRGP_ellipseArc(rect,0,360);

 for(i=0;i<2000000;i++){}
```

```
 do
  {
   SRGP_sampleLocator(&LocMeasure);
   x3=LocMeasure.position.x;
   y3=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x3>=(WIN_X_SIZE-50) || y3<=50);

 StartAngle=(int)(atan2((double)(y2-y1),(double)(x2-x1))*180.0/3.14);
 EndAngle=(int)(atan2((double)(y3-y1),(double)(x3-x1))*180.0/3.14);

 if(StartAngle<0)
  StartAngle=360+StartAngle;

 if(EndAngle<0)
  EndAngle=360+EndAngle;

 if(x2==x3 && y2==y3)
  {
   StartAngle=0;
   EndAngle=360;
  }

 SRGP_setLocatorEchoType(CURSOR);
 SRGP_setInputMode(LOCATOR,EVENT);

 ReloadCanvas();

 SRGP_ellipseArc(rect,StartAngle,EndAngle);

 CreateMenuBar();
 CreateColorPalette();
 SetHint("Επέλεξε το κέντρο και",
        "δύο σημεία του κύκλου");
}

void CreateEllipseArc(int x1,int y1)
{
 int x2=0,y2=0,x3=0,y3=0,x4=0,y4=0,i;
 point pos;
 rectangle rect;
 int StartAngle,EndAngle;

 SaveCanvas();

 pos.x=x1;
 pos.y=y1;

 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_RECT);

 do
  {
   SRGP_sampleLocator(&LocMeasure);
   x2=LocMeasure.position.x;
   y2=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);
```

```
if(x1<=x2 && y1<=y2)
 {
  rect.bottom_left.x=x1;
  rect.bottom_left.y=y1;
  rect.top_right.x=x2;
  rect.top_right.y=y2;
 }
else if(x2<x1 && y2<y1)
 {
  rect.bottom_left.x=x2;
  rect.bottom_left.y=y2;
  rect.top_right.x=x1;
  rect.top_right.y=y1;
 }
else if(x1<=x2 && y2<y1)
 {
  rect.bottom_left.x=x1;
  rect.bottom_left.y=y2;
  rect.top_right.x=x2;
  rect.top_right.y=y1;
 }
else if(x2<x1 && y1<=y2)
 {
  rect.bottom_left.x=x2;
  rect.bottom_left.y=y1;
  rect.top_right.x=x1;
  rect.top_right.y=y2;
 }

SRGP_ellipseArc(rect,0,360);

pos.x=(x1+x2)/2;
pos.y=(y1+y2)/2;

SRGP_setLocatorEchoType(CURSOR);
SRGP_setLocatorEchoRubberAnchor(pos);
SRGP_setLocatorEchoType(RUBBER_LINE);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
  x3=LocMeasure.position.x;
  y3=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x3>=(WIN_X_SIZE-50) || y3<=50);

SRGP_setLineStyle(DOTTED);
SRGP_lineCoord(pos.x,pos.y,x3,y3);
SRGP_setLineStyle(CONTINUOUS);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
  x4=LocMeasure.position.x;
  y4=LocMeasure.position.y;
```

```
 }
while(LocMeasure.button_chord[0]==UP || x4>=(WIN_X_SIZE-50) || y4<=50);

SRGP_setLocatorEchoType(CURSOR);
SRGP_setInputMode(LOCATOR,EVENT);

StartAngle=(int)(atan2((double)(y3-pos.y),(double)(x3-pos.x))*180.0/3.14);
EndAngle=(int)(atan2((double)(y4-pos.y),(double)(x4-pos.x))*180.0/3.14);

if(StartAngle<0)
 StartAngle=360+StartAngle;

if(EndAngle<0)
 EndAngle=360+EndAngle;

if(x3==x4 && y3==y4)
 {
  StartAngle=0;
  EndAngle=360;
 }

ReloadCanvas();

SRGP_ellipseArc(rect,StartAngle,EndAngle);
}

void CreateParaboleArc(int x1,int y1)
{
 int x2=0,y2=0,x3=0,y3=0,x4=0,y4=0,i,y,x,count;
 float a;
 point pos,*spline;
 int StartPoint,EndPoint;
 float d,MinFromStart,MinFromEnd;
 boolean REV;

 SaveCanvas();

 pos.x=x1;
 pos.y=y1;

 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_RECT);

 do
  {
   SRGP_sampleLocator(&LocMeasure);
   x2=LocMeasure.position.x;
   y2=LocMeasure.position.y;
  }
 while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);

 REV=False;

 if(x2<x1 && y2<y1)
  {
   swap(&x2,&x1);
   swap(&y2,&y1);
   REV=True;
```

```
 }
else if(x1<=x2 && y2<y1)
 {
  swap(&y2,&y1);
  REV=True;
 }
else if(x2<x1 && y1<=y2)
 {
  swap(&x2,&x1);
 }

spline=(point *)calloc(x2-x1+1,sizeof(point));
count=0;

a=(y2-y1)/pow(((x2-x1)/2),2);

for(i=x1;i<=x2;i++)
 {
  x=(x1+x2)/2-i;
  if((a*x*x)>(floor(a*x*x)+0.5))
   y=floor(a*x*x)+1;
  else
   y=floor(a*x*x);
  spline[count].x=i;
  if(REV==False)
   spline[count].y=y+y1;
  else
   spline[count].y=y2-y;
  count++;
 }

for(i=0;i<x2-x1;i++)
 SRGP_lineCoord(spline[i].x,spline[i].y,spline[i+1].x,spline[i+1].y);

pos.x=(x1+x2)/2;
pos.y=(y1+y2)/2;

SRGP_setLocatorEchoType(CURSOR);
SRGP_setLocatorEchoRubberAnchor(pos);
SRGP_setLocatorEchoType(RUBBER_LINE);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
  x3=LocMeasure.position.x;
  y3=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x3>=(WIN_X_SIZE-50) || y3<=50);

SRGP_setLineStyle(DOTTED);
SRGP_lineCoord(pos.x,pos.y,x3,y3);
SRGP_setLineStyle(CONTINUOUS);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
```

```
  x4=LocMeasure.position.x;
  y4=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x4>=(WIN_X_SIZE-50) || y4<=50);

StartPoint=0;
MinFromStart=sqrt(pow(spline[0].x-x3,2)+pow(spline[0].y-y3,2));
EndPoint=0;
MinFromEnd=sqrt(pow(spline[0].x-x4,2)+pow(spline[0].y-y4,2));

for(i=1;i<=x2-x1;i++)
 {
  d=sqrt(pow(spline[i].x-x3,2)+pow(spline[i].y-y3,2));
  if(d<MinFromStart)
   {
    MinFromStart=d;
    StartPoint=i;
   }

  d=sqrt(pow(spline[i].x-x4,2)+pow(spline[i].y-y4,2));
  if(d<MinFromEnd)
   {
    MinFromEnd=d;
    EndPoint=i;
   }
 }

SRGP_setLocatorEchoType(CURSOR);
SRGP_setInputMode(LOCATOR,EVENT);

ReloadCanvas();

if(StartPoint>EndPoint)
 swap(&StartPoint,&EndPoint);

for(i=StartPoint;i<EndPoint;i++)
 SRGP_lineCoord(spline[i].x,spline[i].y,spline[i+1].x,spline[i+1].y);
}

void CreateYperboleArc(int x1,int y1)
{
 int x2=0,y2=0,x3=0,y3=0,x4=0,y4=0,Ax=0,Ay=0,i,y,x,count;
 float a;
 point pos,*spline;
 int StartPoint,EndPoint;
 float d,MinFromStart,MinFromEnd;
 QUART q;

 SaveCanvas();

 pos.x=x1;
 pos.y=y1;

 SRGP_waitEvent(-1);
 SRGP_setInputMode(LOCATOR,SAMPLE);
 SRGP_setLocatorEchoRubberAnchor(pos);
 SRGP_setLocatorEchoType(RUBBER_RECT);

 do
  {
```

```
 SRGP_sampleLocator(&LocMeasure);
 x2=LocMeasure.position.x;
 y2=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x2>=(WIN_X_SIZE-50) || y2<=50);

SRGP_setLineStyle(DOTTED);

if(x1<=x2 && y1<=y2)
 {
 q=FIRST;
 SRGP_lineCoord(x1,y1,x1,y2);
 SRGP_lineCoord(x1,y1,x2,y1);
 pos.x=x1;
 pos.y=y1;
 }
else if(x2<x1 && y2<y1)
 {
 q=THIRD;
 swap(&x2,&x1);
 swap(&y2,&y1);
 SRGP_lineCoord(x2,y2,x2,y1);
 SRGP_lineCoord(x2,y2,x1,y2);
 pos.x=x2;
 pos.y=y2;
 }
else if(x1<=x2 && y2<y1)
 {
 q=FOURTH;
 swap(&y2,&y1);
 SRGP_lineCoord(x1,y2,x1,y1);
 SRGP_lineCoord(x1,y2,x2,y2);
 pos.x=x1;
 pos.y=y2;
 }
else if(x2<x1 && y1<=y2)
 {
 q=SECOND;
 swap(&x2,&x1);
 SRGP_lineCoord(x2,y1,x2,y2);
 SRGP_lineCoord(x2,y1,x1,y1);
 pos.x=x2;
 pos.y=y1;
 }

SRGP_setLineStyle(CONTINUOUS);

SRGP_setLocatorEchoType(CURSOR);
SRGP_setLocatorEchoRubberAnchor(pos);
SRGP_setLocatorEchoType(RUBBER_LINE);

for(i=0;i<2000000;i++){}

do
 {
 SRGP_sampleLocator(&LocMeasure);
 Ax=LocMeasure.position.x;
 Ay=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP
```

```
    || Ax>=(WIN_X_SIZE-50) || Ay<=50
    || CheckClick(Ax-pos.x,Ay-pos.y,q)==False);

spline=(point *)calloc(x2-x1,sizeof(point));
count=0;

a=(Ay-pos.y)*(Ax-pos.x);

if(q==THIRD)
 {
  for(i=x1;i<x2;i++)
   {
    x=-x2+i;
    if((a/x)<(floor(a/x)-0.5))
     y=floor(a/x)-1;
    else
     y=floor(a/x);
    spline[count].x=i;

    if((y+y2)<y1)
     spline[count].y=-1;
    else
     spline[count].y=y+y2;
    count++;
   }
 }
else if(q==FIRST)
 {
  for(i=x1+1;i<=x2;i++)
   {
    x=i-x1;
    if((a/x)>(floor(a/x)+0.5))
     y=floor(a/x)+1;
    else
     y=floor(a/x);
    spline[count].x=i;

    if((y+y1)>y2)
     spline[count].y=-1;
    else
     spline[count].y=y+y1;
    count++;
   }
 }
else if(q==SECOND)
 {
  for(i=x1;i<x2;i++)
   {
    x=-x2+i;
    if((a/x)>(floor(a/x)+0.5))
     y=floor(a/x)+1;
    else
     y=floor(a/x);
    spline[count].x=i;

    if((y+y1)>y2)
     spline[count].y=-1;
    else
     spline[count].y=y+y1;
    count++;
```

```
  }
 }
else if(q==FOURTH)
 {
  for(i=x1+1;i<=x2;i++)
   {
    x=i-x1;
    if((a/x)<(floor(a/x)-0.5))
     y=floor(a/x)-1;
    else
     y=floor(a/x);
    spline[count].x=i;

    if((y+y2)<y1)
     spline[count].y=-1;
    else
     spline[count].y=y+y2;
    count++;
   }
 }

for(i=0;i<x2-x1-1;i++)
 {
  if(spline[i].y==-1 || spline[i+1].y==-1)
   continue;
  SRGP_lineCoord(spline[i].x,spline[i].y,spline[i+1].x,spline[i+1].y);
 }

pos.x=(x1+x2)/2;
pos.y=(y1+y2)/2;

SRGP_setLocatorEchoType(CURSOR);
SRGP_setLocatorEchoRubberAnchor(pos);
SRGP_setLocatorEchoType(RUBBER_LINE);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
  x3=LocMeasure.position.x;
  y3=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x3>=(WIN_X_SIZE-50) || y3<=50);

SRGP_setLineStyle(DOTTED);
SRGP_lineCoord(pos.x,pos.y,x3,y3);
SRGP_setLineStyle(CONTINUOUS);

for(i=0;i<2000000;i++){}

do
 {
  SRGP_sampleLocator(&LocMeasure);
  x4=LocMeasure.position.x;
  y4=LocMeasure.position.y;
 }
while(LocMeasure.button_chord[0]==UP || x4>=(WIN_X_SIZE-50) || y4<=50);

StartPoint=0;
```

```
 MinFromStart=sqrt(pow(spline[0].x-x3,2)+pow(spline[0].y-y3,2));
 EndPoint=0;
 MinFromEnd=sqrt(pow(spline[0].x-x4,2)+pow(spline[0].y-y4,2));

 for(i=1;i<=x2-x1-1;i++)
  {
   d=sqrt(pow(spline[i].x-x3,2)+pow(spline[i].y-y3,2));
   if(d<MinFromStart)
    {
     MinFromStart=d;
     StartPoint=i;
    }

   d=sqrt(pow(spline[i].x-x4,2)+pow(spline[i].y-y4,2));
   if(d<MinFromEnd)
    {
     MinFromEnd=d;
     EndPoint=i;
    }
  }

 SRGP_setLocatorEchoType(CURSOR);
 SRGP_setInputMode(LOCATOR,EVENT);

 ReloadCanvas();

 if(StartPoint>EndPoint)
  swap(&StartPoint,&EndPoint);

 for(i=StartPoint;i<EndPoint;i++)
  SRGP_lineCoord(spline[i].x,spline[i].y,spline[i+1].x,spline[i+1].y);
}

void SaveCanvas()
{
 rectangle rect;
 point dest_corner={0,0};

 ScreenCanvas=SRGP_inquireActiveCanvas();
 TempCanvas=SRGP_createCanvas(WIN_X_SIZE,WIN_Y_SIZE);
 SRGP_useCanvas(TempCanvas);

 rect.bottom_left.x=0;
 rect.bottom_left.y=0;
 rect.top_right.x=WIN_X_SIZE;
 rect.top_right.y=WIN_Y_SIZE;

 SRGP_copyPixel(ScreenCanvas,rect,dest_corner);
 SRGP_useCanvas(ScreenCanvas);
}

void ReloadCanvas()
{
 rectangle rect;
 point dest_corner={0,0};

 rect.bottom_left.x=0;
 rect.bottom_left.y=0;
 rect.top_right.x=WIN_X_SIZE;
 rect.top_right.y=WIN_Y_SIZE;
```

```
 SRGP_copyPixel(TempCanvas,rect,dest_corner);

 SRGP_deleteCanvas(TempCanvas);
}

void CreateProgressBar()
{
 CreateRect3D(660,10,780,40,LIGHTGRAY,Down);
}

void UpdateProgressBar(int percent)
{
 char per[10];
 point pos={800,20};
 int NewX;

 NewX=663+(percent*114/100);
 if(OldX!=NewX)
  {
   sprintf(per,"%i %%",percent);

   SRGP_setColor(SKYBLUE);
   SRGP_fillRectangleCoord(790,5,840,45);

   SRGP_setColor(MAROON);
   SRGP_fillRectangleCoord(OldX,13,NewX,37);

   SRGP_text(pos,per);

   OldX=NewX;
  }
 SRGP_setColor(LastColor);
}

void SetHint(char *line1,char *line2)
{
 point pos1={650,30};
 point pos2={650,10};

 ClearHint();
 SRGP_setColor(SRGP_BLACK);
 SRGP_text(pos1,line1);
 SRGP_text(pos2,line2);
 SRGP_setColor(LastColor);
}

void ClearHint()
{
 SRGP_setColor(SKYBLUE);
 SRGP_fillRectangleCoord(640,5,840,45);
 SRGP_setColor(LastColor);
}

void swap(int *a,int *b)
{
 int temp;

 temp=*a;
 *a=*b;
```

```
 *b=temp;
}

boolean CheckClick(int dx,int dy,QUART q)
{
 if(q==FIRST && (dx<0 || dy<0))
  return(False);
 else if(q==SECOND &&(dx>0 || dy<0))
  return(False);
 else if(q==THIRD &&(dx>0 || dy>0))
  return(False);
 else if(q==FOURTH &&(dx<0 || dy>0))
  return(False);
 else
  return(True);
}

#include <srgp.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define APL_NAME  "Εφαρμογή Γραφικών"
#define WIN_X_SIZE 650
#define WIN_Y_SIZE 650
#define COLORS                  24          /* Total Colors = 2^COLORS */
#define ENABLE_TRACE            1

#define SKYBLUE   2
#define LIGHTBLUE 3
#define BLUE      4
#define CYAN      5
#define GREEN     6
#define SEAGREEN  7
#define YELLOW    8
#define RED       9
#define ORANGE    10
#define MAROON    11
#define GRAY      12
#define LIGHTGRAY 13
#define GOLD      14
#define CORAL     15

#define BUTTONS   8
#define COLOR_BUTTONS 16

typedef enum{Up,Down}FORM;
typedef enum{COPY,LINE,CIRCLE,ELLIPSE,PARABOLE,YPERBOLE,NONE}DESIGN;
typedef enum{FIRST,SECOND,THIRD,FOURTH}QUART;

typedef struct
        {
         boolean form;
         int x;
         int y;
         int dx;
         int dy;
         int tag;
         int color;
        }Button;
```

```
void Initialize(void);
void CreateMenuBar(void);
void CreatePopUpMenu(void);
void ClearPopUpMenu(void);
void CreateColorPalette(void);
void CreateRect3D(int x1,int y1,int x2,int y2,int color,FORM form);
void CreateButton(Button btn);
void FlushButton(Button btn);
void LoadCommonColors(void);
void OpenFile(char *fname);
void SaveFile(char *fname);
void CreateDialog(char *caption);
void ClearDialog(void);
void ClearPanel(void);
void CopyRegion(int x1,int y1);
void CreateLine(int x1,int y1);
void CreateCircleArc(int x1,int y1);
void CreateEllipseArc(int x1,int y1);
void CreateParaboleArc(int x1,int y1);
void CreateYperboleArc(int x1,int y1);
void SaveCanvas(void);
void ReloadCanvas(void);
void CreateProgressBar(void);
void UpdateProgressBar(int percent);
void SetHint(char *line1,char *line2);
void ClearHint();
void swap(int *a,int *b);
boolean CheckClick(int dx,int dy,QUART q);

Button MenuButton,OkBtn,CancelBtn;
Button button[BUTTONS];
Button color_button[COLOR_BUTTONS];
locator_measure LocMeasure;
int LastColor;
boolean MenuFlag,DialogFlag,OpenFlag,SaveFlag;
DESIGN DesignFlag;
char FileName[30];
canvasID ScreenCanvas,TempCanvas;
int OldX=663;
```

NAME

glAccum - operate on the accumulation buffer

C SPECIFICATION

void **glAccum**(GLenum *op*,
GLfloat *value*)

PARAMETERS

op Specifies the accumulation buffer operation. Symbolic constants **GL_ACCUM**, **GL_LOAD**, **GL_ADD**, **GL_MULT**, and **GL_RETURN** are accepted.

value Specifies a floating-point value used in the accumulation buffer operation. *op* determines how *value* is used.

DESCRIPTION

The accumulation buffer is an extended-range color buffer.

Images are not rendered into it. Rather, images rendered into one of the color buffers are added to the contents of the accumulation buffer after rendering. Effects such as antialiasing (of points, lines, and polygons), motion blur, and depth of field can be created by accumulating images generated with different transformation matrices.

Each pixel in the accumulation buffer consists of red, green, blue, and alpha values. The number of bits per component in the accumulation buffer depends on the implementation. You can examine this number by calling **glGetIntegerv** four times, with arguments **GL_ACCUM_RED_BITS**, **GL_ACCUM_GREEN_BITS**, **GL_ACCUM_BLUE_BITS**, and **GL_ACCUM_ALPHA_BITS**. Regardless of the number of bits per component, the range of values stored by each component is [-1, 1]. The accumulation buffer pixels are mapped one-to-one with frame buffer pixels.

glAccum operates on the accumulation buffer. The first argument, *op*, is a symbolic constant that selects an accumulation buffer operation. The second argument, *value*, is a floating-point value to be used in that operation. Five operations are specified: **GL_ACCUM**, **GL_LOAD**, **GL_ADD**, **GL_MULT**, and **GL_RETURN**.

All accumulation buffer operations are limited to the area of the current scissor box and applied identically to the red, green, blue, and alpha components of each pixel. If a **glAccum** operation results in a value outside the range [-1, 1], the contents of an accumulation buffer pixel component are undefined.

The operations are as follows:

GL_ACCUM Obtains R, G, B, and A values from the buffer currently selected for reading (see **glReadBuffer**). Each component value is divided by 2n-1, where n is the number of bits allocated to each color component in the currently selected buffer. The result is a floating-point value in the range [0, 1], which is multiplied by *value* and added to the corresponding pixel component in the accumulation buffer, thereby updating the accumulation buffer.

GL_LOAD Similar to **GL_ACCUM**, except that the current value in the accumulation buffer is not used in the calculation of the new value. That is, the R, G, B, and A values from the currently selected buffer are divided by 2n-1, multiplied by *value*, and then stored in the corresponding accumulation buffer cell, overwriting the current value.

GL_ADD Adds *value* to each R, G, B, and A in the accumulation buffer.

GL_MULT Multiplies each R, G, B, and A in the accumulation buffer by *value* and returns the

scaled component to its corresponding accumulation buffer location.

GL_RETURN Transfers accumulation buffer values to the color buffer or buffers currently selected for writing. Each R, G, B, and A component is multiplied by *value*, then multiplied by 2n-1, clamped to the range [0,2n-1], and stored in the corresponding display buffer cell. The only fragment operations that are applied to this transfer are pixel ownership, scissor, dithering, and color writemasks.

To clear the accumulation buffer, call **glClearAccum** with R, G, B, and A values to set it to, then call **glClear** with the accumulation buffer enabled.

NOTES

Only pixels within the current scissor box are updated by a **glAccum** operation.

ERRORS

GL_INVALID_ENUM is generated if *op* is not an accepted value.

GL_INVALID_OPERATION is generated if there is no accumulation buffer.

GL_INVALID_OPERATION is generated if **glAccum** is executed between the execution of **glBegin** and the corresponding execution of **glEnd**.

ASSOCIATED GETS

glGet with argument **GL_ACCUM_RED_BITS**
glGet with argument **GL_ACCUM_GREEN_BITS**
glGet with argument **GL_ACCUM_BLUE_BITS**
glGet with argument **GL_ACCUM_ALPHA_BITS**

SEE ALSO

glBlendFunc, **glClear**, **glClearAccum**, **glCopyPixels**, **glGet**, **glLogicOp**, **glPixelStore**, **glPixelTransfer**, **glReadBuffer**, **glReadPixels**, **glScissor**, **glStencilOp**

NAME

glColorPointer - define an array of colors

C SPECIFICATION

void **glColorPointer**(GLint *size*,
GLenum *type*,
GLsizei *stride*,
const GLvoid **pointer*)

PARAMETERS

size Specifies the number of components per color. Must be 3 or 4.

type Specifies the data type of each color component in the array. Symbolic constants **GL_BYTE**, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_UNSIGNED_SHORT**,

GL_INT, **GL_UNSIGNED_INT**, **GL_FLOAT**, and **GL_DOUBLE** are accepted.

stride Specifies the byte offset between consecutive colors. If *stride* is 0, (the initial value), the colors are understood to be tightly packed in the array.

pointer Specifies a pointer to the first component of the first color element in the array.

DESCRIPTION

glColorPointer specifies the location and data format of an array of color components to use when rendering. *size* specifies the number of components per color, and must be 3 or 4. *type* specifies the data type of each color component, and *stride* specifies the byte stride from one color to the next allowing vertexes and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations; see **glInterleavedArrays**.)

When a color array isspecified, *size*, *type*, *stride*, and *pointer* are saved as client-side state.

To enable and disable the color array, call **glEnableClientState** and **glDisableClientState** with the argument **GL_COLOR_ARRAY**. If enabled, the color array is used when **glDrawArrays**, **glDrawElements**, or **glArrayElement** is called.

NOTES

glColorPointer is available only if the GL version is 1.1 or greater.
The color array is initially disabled and isn't accessed when **glArrayElement**, **glDrawArrays**, or **glDrawElements** is called.

Execution of **glColorPointer** is not allowed between the execution of **glBegin** and the corresponding execution of **glEnd**, but an error may or may not be generated. If no error is generated, the operation is undefined.

glColorPointer is typically implemented on the client side.

Color array parameters are client-side state and are therefore not saved or restored by **glPushAttrib** and **glPopAttrib**. Use **glPushClientAttrib** and **glPopClientAttrib** instead.

ERRORS

GL_INVALID_VALUE is generated if *size* is not 3 or 4.

GL_INVALID_ENUM is generated if *type* is not an accepted value.

GL_INVALID_VALUE is generated if *stride* is negative.

ASSOCIATED GETS

glIsEnabled with argument **GL_COLOR_ARRAY**

glGet with argument **GL_COLOR_ARRAY_SIZE**
glGet with argument **GL_COLOR_ARRAY_TYPE**
glGet with argument **GL_COLOR_ARRAY_STRIDE**
glGetPointerv with argument **GL_COLOR_ARRAY_POINTER**

SEE ALSO

glArrayElement, **glDrawArrays**, **glDrawElements**, **glEdgeFlagPointer**, **glEnable**, **glGetPointerv**, **glIndexPointer**, **glInterleavedArrays**, **glNormalPointer**, **glPopClientAttrib**, **glPushClientAttrib**, **glTexCoordPointer**, **glVertexPointer**