ΤΕΙ ΜΕΣΟΛΟΓΓΙΟΥ

ΤΜΗΜΑ ΤΗΛΕΠ/ΚΩΝ ΣΥΣΤΗΜΑΤΩΝ & ΔΙΚΤΥΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ:

Αξιολόγηση Πρωτοκόλλων Web-TV σε Δίκτυα IPV6

Στοιχεία Φοιτητή :

Τσατσιμάς Νικόλαος

ΑΜ:0465

Επιβλέποντες Καθηγητές :

Μαριάτος Ευάγγελος

ΝΑΥΠΑΚΤΟΣ 2012

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΜΕΣΟΛΟΓΓΙΟΥ

ΤΜΗΜΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΟΚΤΩΒΡΗΣ 2012

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κύριο Μαριάτο Ευάγγελο για την δυνατότητα που μου έδωσε να εκτελέσω την διεξαγωγή της πτυχιακής μου αλλά επίσης για το ικανοποιητικό για μένα θέμα που ανέλαβα έτσι ώστε να διερευνήσω ακόμα περισσότερο τον τομέα των δικτύων στο πως λειτουργούν. Ακόμα θα ήθελα να ευχαριστήσω τους γονείς μου για την οικονομική αλλά και πνευματική στήριξη που με βοήθησαν εδώ και τέσσερα χρόνια να εκτελώ τις υποχρεώσεις μου ως ένας άξιος φοιτητής του τμήματος στο οποίο φοίτησα

ΠΕΡΙΕΧΟΜΕΝΑ

Παράρτημα Α: Βιβλιογραφία και πηγές απο το διαδίκτυο

Παράρτημα Β: Κανόνες της γλώσσας C

ΠΕΡΙΛΗΨΗ

Η ακόλουθη πτυχιακή εργασία μας περιγράφει τις διαδικασίες που χρειάζονται να ξέρουμε για την μετάδοση μιας εικόνας απο έναν υπολογιστή στον άλλον. Ιεραρχικά ξεκινά να μελετά τον τρόπο που μεταδίδουμε τα δεδομένα απο τον ένα υπολογιστή στον άλλο δηλαδή το είδος του δικτύου , που στην περίπτωση μας το δίκτυο αυτό ονομάζεται multicast . Όπου αναλύουμε τι σημαίνει multicast ,την βασική του αρχιτεκτονική και την διευθυνσιοδότηση . Παρακάτω αναλύουμε το διαδικτυακό πρωτόκολλο στο οποίο τρέχει το υπάρχον multicast δίκτυο δηλαδή το ipv6 . Αναλύουμε την αρχιτεκτονική του τα πλεονεκτήματα του και την διευθυνσιοδότηση του . Αφού αναλύσαμε ήδη το μέσω με το οποίο θα μεταδώσουμε την εικόνα αμέσως μετά αναλύουμε το είδος της εικόνας που θα μεταδώσουμε που στην περίπτωση μας είναι η διαμόρφωση BMP και αναλύουμε πως σε αυτή την περίπτωση διαμοιράζονται τα χρώματα τελικά σας δίνουμε τον κώδικα ώστε να μπορείτε να πειραματιστείτε και να δοκιμάσετε από μόνη σας το όλο εγχείρημα αφού σας περιγράφουμε γραμμή γραμμή τι κάνουμε .

ABSTRACT

The following graduation describes the procedures we needed to know to transmit an image from one computer to another. Hierarchical begins to study how transmit data from a computer to another ie the type of network, which in our case the network is called multicast. When we analyze what is multicast, the basic architecture of and addressing. Below we analyze the Internet protocol which runs the existing multicast network that is the ipv6. We analyze the architecture of, the advantages and addressing . Having already analyzed by means of which feeds the image . We analyze the type of image witch will send, in our case is the formation BMP and analyze how the colors are shared.Finally give you the code to experiment and try by yourself the whole thing as we describe what we do line by line.

MULTICASTING

1.1 Πως μπορούμε να μεταφέρουμε πληροφορία από ένα Η/Υ σε έναν άλλο (multicasting)

Για την επικοινωνία δύο υπολογιστικών συστημάτων στο ίδιο δίκτυο μπορεί να χρησιμοποιηθεί το πρωτόκολλο multicast . Με το εξής πρωτόκολλο μπορούμε να μεταφέρουμε πληροφορία (πακέτα udp) από τον έναν υπολογιστή στον άλλον ή ακόμα και από έναν σε πολλούς υπολογιστές οι οποίοι ανήκουν στο ίδιο δίκτυο ,σε αυτή την περίπτωση κάθε υπολογιστής «ακούει» τα πακέτα όλων των άλλων υπολογιστών που αποστέλλονται στο δίκτυο και αναζητά τα πακέτα που απευθύνονται σε αυτόν .Όταν ο υπολογιστής εντοπίσει το πακέτο που απευθύνεται σε αυτόν διακόπτει την μετάδοση των πακέτων και παίρνει το πακέτο του και το περνάει στο λειτουργικό του σύστημα για να το επεξεργαστεί. Με το multicasting ο αποστολέας υπολογιστής στέλνει ένα σύνολο πακέτων που απευθύνονται προς ένα σύνολο υπολογιστών οι οποίοι ανήκουν σε ένα συγκεκριμένο σύνολο multicasting διευθύνσεων. Έτσι με αυτό τον τρόπο όσοι υπολογιστές ενδιαφέρονται να συμμετέχουν σε μία multicasting επικοινωνία προγραμματίζονται σε αυτές τις διευθύνσεις έτσι ώστε να ακούν τα πακέτα που έχουν τις διευθύνσεις τους .

1.2 Ποιά είναι τα πλεονεκτήματα του multicasting

Το multicasting βελτιστοποιεί την απόδοση του δικτύου .Επειδή αποστέλλονται μόνο σε ένα σύνολο δεδομένων .Το multicasting δεσμεύει ένα εύρος ζώνης στο δίκτυο και ελαχιστοποιεί τον πλεονασμό μετάδοσης των δεδομένων . Επίσης παρέχει μεγαλύτερη αποδοτικότητα ελέγχοντας την μετάδοση των δεδομένων στο δίκτυο και μειώνοντας το φορτίο στις συσκευές του δικτύου. Οι χρήστες του δικτύου είναι σε θέση να «ακούσουν» ή όχι την multicast διεύθυνση ,και έτσι τα πακέτα δεδομένων αποστέλλονται μόνο όπου απαιτείται . Επιπλέον το multicasting προσαρμόζεται κατάλληλα σε κάθε δίκτυο διαφορετικού μεγέθους ,αλλά ιδιαίτερα είναι ευέλικτο σε περιβάλλοντα δικτύων ευρείας περιοχής (wan) . Το multicasting προσφέρει στούς χρήστες που είναι εγκαταστημένοι σε διαφορετικές περιοχές πρόσβαση σε αρχεία δεδομένων μεγάλου μεγέθους , όπως εικόνα , μια ταινία ή μια ζωντανή παρουσίαση

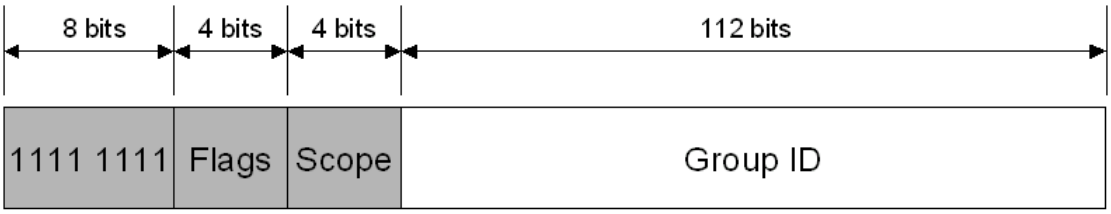
χωρίς να δεσμεύει μεγάλο εύρος ζώνης ή να απαιτείται broadcasting σε όλους τους χρήστες του διαδικτύου.

1.3 Πως λειτουργούν οι multicast διευθύνσεις

Επειδή οι multicast διευθύνσεις αναγνωρίζουν μια συνδιάλεξη μετάδοσης παρά μια διεύθυνση λήψης ή αποστολής , όλοι οι αποδέκτες σε μια multicast ομάδα αναγνωρίζονται από μια απλή IP διεύθυνση . Στη παρακάτω παράγραφο περιγράφεται η δομή των multicast ip mac διευθύνσεων .

1.4 Πως οργανώνονται οι ipv6 multicast διευθύνσεις

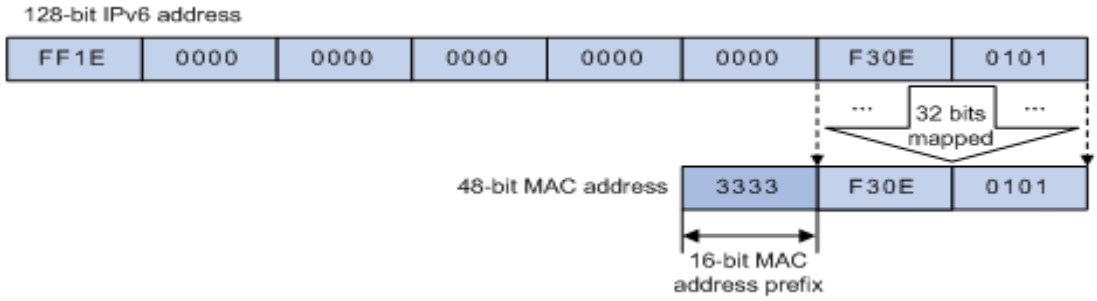
Μία ipv6 multicast διεύθυνση αποτελεί μια ipv6 διεύθυνση , η οποία διαθέτει τα εξής bit υψηλής προτεραιότητας στην πρώτη οκτάδα 11111111 στο δυαδικό σύστημα , ή FF στο δεκαεξαδικό σύστημα . Στη δεύτερη οκτάδα τα πρώτα τρία bit δεσμεύονται και είναι όλα 0. Το επόμενο bit στην οκτάδα προσδιορίζει την διάρκεια ζωής της multicast διεύθυνσης . Η multicast διεύθυνση που έχει το 0 στο bit της παραμέτρου της διάρκειας ζωής είναι μόνιμη , ενώ η multicast διεύθυνση που έχει το 1 στο bit της παραμέτρου της διάρκειας ζωής είναι προσωρινή . Τα επόμενα 4 bit στην οκτάδα προσδιορίζουν την εμβέλεια της multicast διεύθυνσης . Η εμβέλεια καθορίζεται ως εξής 1= εμβέλεια τοπικού κόμβου , 2=εμβέλεια τοπικής σύνδεσης , 5=τοπική εμβέλεια ,8=εμβέλεια οργανισμού , E=παγκόσμια εμβέλεια . Για παράδειγμα η multicast διεύθυνση με πρόθεμα FF02 είναι μόνιμη multicast διεύθυνση με εμβέλεια τοπικής σύνδεσης. Καθορίζοντας την εμβέλεια είναι δυνατόν να περιοριστεί η διαδρομή του πακέτου με ένα πιο αξιόπιστο τρόπο από ότι στο ipv4. Οι δρομολογητές πρέπει να γνωρίζουν πότε οι συνδέσεις φτάνουν στα όρια μιας εμβέλειας για να λειτουργούν σωστά , γεγονός απλό για την εμβέλεια τοπικού κόμβου και τοπικής σύνδεσης. Αλλά απαιτεί ειδική παραμετροποίηση όταν η εμβέλεια είναι τοπική και οργανισμού .



Σχήμα 1. Δομή ipv6 multicast Διευθύνσεων

1.5 Πως οι ipv6 multicast διευθύνσεις αντιστοιχίζονται με τις mac multicast διευθύνσεις

Στο ipv6 η κεφαλίδα της mac multicast διεύθυνσης είναι πάντοτε 33.33 και τα εναπομείναντα 32 bit αντιστοιχίζονται απευθείας με τα 32 λιγότερο σημαντικά bit της ipv6 multicast διεύθυνσης . Η ipv6 διεύθυνση διαθέτει επιπλέον 88 bit που είναι αδιάφορα όταν υλοποιείται η αντιστοίχιση με τις mac διευθύνσεις . Αυτά τα επιπλέον bit προσδιορίζουν τις 2(στην 88) ipv6 διευθύνσεις που χρησιμοποιούν τις ίδιες mac διευθύνσεις . Ένας υπολογιστής είναι σε θέση να δεχτεί οποιοδήποτε multicasting ,φιλτράροντας τα επιπλέον multicast πακέτα που αποστέλλονται με την ίδια mac multicast διεύθυνση . Συνήθως τα πρωτόκολλα υψηλότερου επιπέδου προσδιορίζουν ποιά πακέτα είναι της κάθε εφαρμογής.



Σχήμα 2. Αντιστοίχιση μιας IPV6 MULTICAST διεύθυνσης με μια MAC

1.6 Πως υλοποιείται η διαχείριση των multicast δεδομένων

Το multicasting είναι σχετικά απλό όταν υλοποιείται σε ένα τμήμα του δικτύου. Ο αποστολέας καθορίζει μια συγκεκριμένη multicast διεύθυνση προορισμού και ο

οδηγός της συσκευής μετατρέπει την διεύθυνση αυτή στην αντίστοιχη mac διεύθυνση και αποστέλλει το πακέτο στο δίκτυο. Οι αποδέκτες στο δίκτυο πρέπει να υποδείξουν αν επιθυμούν να παραλάβουν τα multicast δεδομένα. Εντούτοις , η πολυπλοκότητα εμφανίζεται όταν το multicasting επεκτείνεται πέρα από ένα δίκτυο και τα multicast πακέτα διέρχονται μέσα από δρομολογητές . Όλες οι συσκευές που συμμετέχουν στο multicasting όπως οι εξυπηρετητές , οι υπολογιστές , οι δρομολογητές και οι μεταγωγοί θα πρέπει να συντονίσουν τις λειτουργίες τους. Η αποκατάσταση της διαδρομής μεταξύ των συσκευών αποστολής και προορισμού και η προώθηση των multicast δεδομένων μέσω του δικτύου είναι θέματα που πρέπει να μελετηθούν για την διαχείριση της multicast μετάδοσης δεδομένων . Η multicast μετάδοση δεδομένων υλοποιείται από την πηγή προς το σύνολο των multicast συσκευών με την έννοια ενός δέντρου διανομής , το οποίο συνδέει το σύνολο των υπολογιστών . Υπάρχει ένα σύνολο διαφορετικών multicast πρωτοκόλλων δρομολόγησης , τα οποία χρησιμοποιούν διαφορετικές τεχνικές κατασκευής των δέντρων αυτών αλλά πριν τα multicast δεδομένα μεταδοθούν στο δίκτυο ,οι δρομολογητές απαιτείται να γνωρίζουν ποίοι υπολογιστές επιθυμούν να συμπεριληφθούν στην multicast ομάδα . Υπάρχει ένας αριθμός πρωτοκόλλων που καθορίζουν την multicast μετάδοση μεταξύ των δικτύων ,όμως το πρώτο βήμα της διαδικασίας αυτής είναι να προσδιοριστούν οι αποδέκτες των multicast δεδομένων.

1.7 IPV6 multicast listener discovery (MLD)

Το MLD ελέγχει αυτόματα τη ροή των δεδομένων σε ένα δίκτυο χρησιμοποιώντας multicast μονάδες δημιουργίας μηνυμάτων ερώτησης και multicast υπολογιστές. Η μονάδα δημιουργίας μηνυμάτων ερώτησης αποτελεί την συσκευή εκείνη του δικτύου ,η οποία αποστέλλει μηνύματα ερωτήσεων για να εντοπίσει ποιες συσκευές του δικτύου επιθυμούν να ενταχθούν σε μια δεδομένη multicast ομάδα. Ένας υπολογιστής αποτελεί ένα αποδέκτη ο οποίος μπορεί να αποστείλει μηνύματα αναφοράς για να πληροφορήσει την μονάδα δημιουργίας μηνυμάτων ερώτησης για την ένταξη του στην multicast ομάδα. Οι μονάδες δημιουργίας μηνυμάτων ερώτησης και οι υπολογιστές χρησιμοποιούν τις MLD αναφορές για να ενταχθούν ή να εγκαταλείψουν μια multicast ομάδα και για να ξεκινήσουν να παραλαμβάνουν multicast δεδομένα

- **Το πρωτόκολλο MLD διαθέτει τρεις τύπους μηνυμάτων :**

Μηνύματα ερώτησης –αυτά περιλαμβάνουν :

Μηνύματα γενικών ερωτήσεων ,

Μηνύματα ειδικών ομάδων,

Μηνύματα ειδικών multicast διευθύνσεων.

- **Μηνύματα αναφοράς**
- **Μηνύματα ολοκλήρωσης**

Όταν το πρωτόκολλο MLD αποστέλλει μαι γενική ερώτηση , το πεδίο της multicast διεύθυνσης γίνεται 0 και η γενική ερώτηση εντοπίζει ποιές multicast διευθύνσεις έχουν αποδέκτες στην συγκεκριμένη σύνδεση . Οι ειδικές ερωτήσεις ομάδας και οι ειδικές ερωτήσεις multicast διεύθυνσης είναι οι ίδιες –η διεύθυνση ομάδας αποτελεί μια multicast διεύθυνση . Τα MLD μηνύματα αναφοράς έχουν το δικό τους πεδίο multicast διεύθυνσης , το οποίο τοποθετείται στην ειδική IPV6 multicast διεύθυνση ,την οποία ο αποστολέας «ακούει» . Τα MLD μηνύματα ολοκλήρωσης έχουν το δικό τους πεδίο multicast διεύθυνσης , το οποίο τοποθετείται στην ειδική IPV6 multicast διεύθυνση , την οποία δεν «ακούν» πλέον . Το MLDv1 σταματά για περίπου δύο δευτερόλεπτα ,όταν οι υπολογιστές της ομάδας επιθυμούν να εγκαταλείψουν την multicast συνδιάλεξη. Όταν ένας υπολογιστής χρησιμοποιώντας το πρωτόκολλο MLDv1 αποστέλλει ένα μήνυμα εγκατάλειψης ,ο δρομολογητής αποστέλλει μηνύματα ερώτησης για να ελέγξει αν ο υπολογιστής ήταν ο τελευταίος MLDv1 που συνδέθηκε στην ομάδα , πριν διακόψει την δρομολόγηση των δεδομένων . Αντίστοιχα το MLDv2 υποστηρίζει το φιλτράρισμα πηγής . Το mld καθιστά ικανούς τους μεταγωγείς να δρομολογούν τα multicast πακέτα προς τους υπολογιστές που επιθυμούν να παραλάβουν τα δεδομένα , παρά να τα αποστείλουν σε όλες τις θύρες τους ανεξάρτητα αν αυτές επιθυμούν ή οχι να τα παραλάβουν .

1.8 Πως Υλοποιείται η Δρομολόγηση των multicast Πακέτων

Επειδή μια multicast διεύθυνση αναγνωρίζει μόνο μια συγκεκριμένη συνδιάλεξη μετάδοσης και όχι ένα συγκεκριμένο προορισμό, η δρομολόγηση των multicast

δεδομένων είναι περισσότερο πολύπλοκη από την δρομολόγηση των unicast δεδομένων. Όταν ο αριθμός των αποδεκτών των multicast δεδομένων είναι μεγάλος και εφόσον η πηγή δεν απαιτείται να γνωρίζει όλες τις διευθύνσεις των αποδεκτών, οι δρομολογητές του δικτύου θα πρέπει να μεταφράσουν τις multicast διευθύνσεις στις διευθύνσεις των αποδεκτών υπολογιστών . Η βασική αρχή της multicast δρομολόγησης στηρίζεται στο γεγονός ότι οι δρομολογητές πρέπει να αλληλεπιδρούν μεταξύ τους για να ανταλλάσσουν πληροφορίες σχετικά με τους γειτονικούς τους δρομολογητές. Για να μεταδοθούν τα multicast δεδομένα , οι προκαθορισμένοι δρομολογητές απαιτείται να δημιουργήσουν δέντρα διανομής και να συνδέσουν όλους τους αποδέκτες της multicast ομάδας . Τα δέντρα διανομής προσδιορίζουν την διαδρομή απο την πηγή προς κάθε αποδέκτη της multicast ομάδας . Υπάρχει ένας αριθμός διαφορετικών τύπων δέντρων διανομής , όμως οι δύο πιο σημαντικοί τύποι είναι τα πηγαία δέντρα και τα διαμοιραζόμενα δέντρα ή κεντρικοποιημένα δέντρα .Τα πηγαία δέντρα επιλέγουν την συντομότερη διαδρομή απο την πηγή μέχρι τους αποδέκτες . Τα πηγαία δέντρα δημιουργούν πολλαπλά δέντρα διανομής , τα οποία προέρχονται από τα υποδίκτυα που είναι συνδεδεμένα απευθείας με την πηγή.Τα διαμοιραζόμενα ή κεντρικοποιημένα δέντρα χρησιμοποιούν κέντρα διανομής και δημιουργούν ένα μόνο δέντρο το οποίο διαμοιράζεται σε όλους τους αποδέκτες της ομάδας . Στην περίπτωση του διαμοιρασμένου δέντρου η αποστολή και λήψη των multicast δεδομένων υλοποιείται μέσω της ίδιας διαδρομής ανεξάρτητα από τις πηγές των δεδομένων.

1.9 Πρωτόκολλα δρομολόγησης multicast

Τα πρωτόκολλα δρομολόγησης διαχειρίζονται την ανταλλαγή πληροφοριών μεταξύ των δρομολογητών και είναι υπεύθυνα για την δημιουργία δέντρων και την δρομολόγηση των multicast πακέτων.Υπάρχειι ένα πλήθος από διαφορετικά πρωτόκολλα δρομολόγησης , όμως γενικότερα όλα αυτά ακολουθούν μια από τις δύο βασικές προσεγγίσεις – του πυκνού τρόπου ή του διάσπαρτου τρόπου δρομολόγησης

1.10 Πρωτόκολλα πυκνού τρόπου δρομολόγησης

Τα πρωτόκολλα πυκνού τρόπου δρομολόγησης βασίζονται στην λογική ότι υπάρχει ένα πλήθος από multicast ομάδες πυκνά κατανεμημένες σε ένα δίκτυο. Λόγω του

γεγονότος αυτού , τα πρωτόκολλα αυτά διοχετεύουν περιοδικά στο δίκτυο multicast δεδομένα για την αποκατάσταση και διατήρηση του δέντρου διανομής. Τα πρωτόκολλα πυκνού τρόπου δρομολόγησης είναι κατάλληλα σε περιβάλλοντα όπου υπάρχει ένα πλήθος από υπολογιστές που επιθυμούν ή πρέπει να λάβουν τα multicast δεδομένα και το εύρος ζώνης υποστηρίζει την μετάδοση των multicast δεδομένων στο δίκτυο.

- **Protocol independent multicast dense mode (PIM-DM)**

Το ανεξάρτητο πρωτόκολλο πολυεκπομπής πυκνού τρόπου είναι κατάλληλο σε περιπτώσεις όπου υφίσταται ένα πλήθος από υπολογιστές πυκνά τοποθετημένους σε κάθε multicast ομάδα . Μεταδίδει πακέτα προς όλους τους δρομολογητές στο δίκτυο και απομονώνει αυτούς που δεν έχουν υπολογιστές που επιθυμούν να παραλάβουν multicast δεδομένα . Η αναφορά του ως ανεξάρτητο πρωτόκολλο καθορίζεται από την δυνατότητα του να να χρησιμοποιεί ως περιεχόμενο τον κάθε πίνακα unicast χωρίς να χρειάζεται να δημιουργήσει και να οργανώσει τον δικό του ξεχωριστό πίνακα δρομολόγησης.

1.11 Πρωτόκολλα διάσπαρτου τρόπου δρομολόγησης

Τα πρωτόκολλα διάσπαρτου τρόπου δρομολόγησης βασίζονται στην λογική ότι οι υπολογιστές της multicast ομάδας που αναμένουν να παραλάβουν multicast δεδομένα είναι διάσπαρτα κατανεμημένοι σε ένα δίκτυο και το διαθέσιμο εύρος ζώνης δεν είναι απαραίτητο μεγάλο . Λόγω του ότι οι υπολογιστές της ομάδας εκτείνονται διάσπαρτοι σε ολόκληρο το δίκτυο ,η μετάδοση των multicast δεδομένων καταλαμβάνει μεγάλο εύρος ζώνης και μπορεί να δημιουργήσει προβλήματα απόδοσης. Επομένως τα πρωτόκολλα διάσπαρτου τρόπου δρομολόγησης είναι πιο επιλεκτικά στον τρόπο μετάδοσης των multicast δεδομένων . Τα πρωτόκολλα αυτά ξεκινούν από άδεια δέντρα διανομής και προσθέτουν κλώνους όταν λαμβάνουν αιτήσεις συμμετοχής.

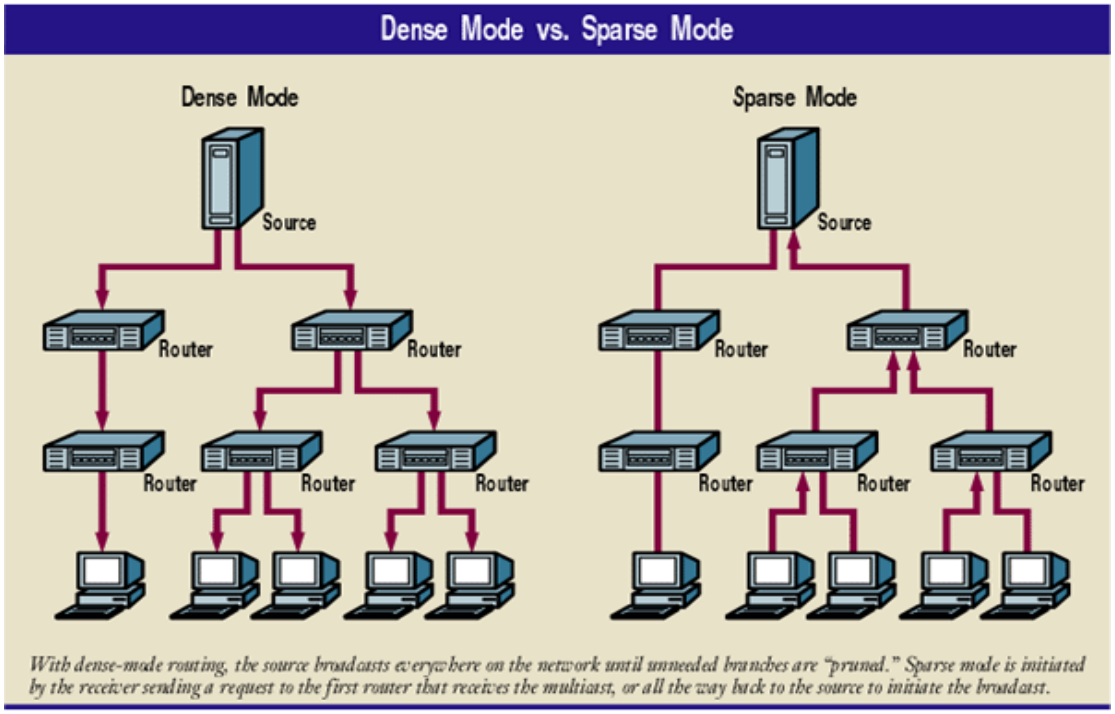
- **Protocol independent sparse mode (PM-SM)**

Το ανεξάρτητο πρωτόκολλο πολυεκπομπής διάσπαρτου τρόπου έχει σχεδιαστεί για περιβάλλοντα όπου οι αποδέκτες είναι ευρέως κατανεμημένοι. Το PIM-SM χρησιμοποιεί ένα σημείο συγκέντρωσης από όπου οι αποστολείς αποστέλλουν τις

πληροφορίες τους και οι αποδέκτες τις αιτήσεις τους . Επομένως όταν ένας αποδέκτης επιθυμεί να παραλάβει multicast δεδομένα αυτός καταχωρείται στο σημείο συγκέντρωσης και μόλις τα δεδομένα ξεκινήσουν να μεταδίδονται από τον αποστολέα το σημείο συγκέντρωσης αναμεταδίδει τα δεδομένα . Οι δρομολογητές αυτόματα βελτιστοποιούν την διαδρομή απαλλάσσοντας την από τα περιττά άλματα. Λόγω του ότι το PIM-SM αποτελεί ένα ανεξάρτητο πρωτόκολλο αυτό χρησιμοποιεί τα δεδομένα που περιέχονται στο υφιστάμενο πρωτόκολλο unicast δρομολόγησης .

1.12 Διαλειτουργικότητα

Τα πρωτόκολλα PIM-DM , PIM-SM μπορούν να λειτουργήσουν την ίδια χρονική στιγμή σε διαφορετικές διασυνδέσεις. Με την δρομολόγηση πυκνού τρόπου δρομολόγησης η πηγή εκπέμπει ευρέως δεδομένα παντού στο δίκτυο και απομονώνει οτιδήποτε ενημερώνει ότι δεν επιθυμεί να παραλάβει τα δεδομένα αυτά . Με την δρομολόγηση διάσπαρτου τρόπου δρομολόγησης οι ενδιαφερόμενοι αποδέκτες κάνουν αίτηση στα δεδομένα από την πηγή ή τον πρώτο δρομολογητή που λαμβάνει multicast δεδομένα .



Σχήμα 3. Πρωτόκολλο Πυκνού τρόπου Δρομολόγησης έναντι Διάσπαρτου τρόπου Δρομολόγησης

IPV6

2.1 Τι είναι το ipv6

Το πρωτόκολλο αυτό αποτελεί την βελτιωμένη έκδοση του υπάρχοντος διαδεδομένου πιο πολύ internet protocol του ipv4 . Το ip είναι υπεύθυνο για την αποστολή των δεδομένων από έναν υπολογιστή σε έναν άλλο σε ένα δίκτυο και είναι γνωστό ως ασυνδεσμικό πρωτόκολλο , διότι δεν υφίσταται μόνιμη σύνδεση μεταξύ των δύο επικοινωνούντων συσκευών . Επομένως όταν μεταδίδεται ένα μήνυμα μέσω του ip αυτό μορφοποιείται σε πακέτα τα οποία μπορούν να μεταδοθούν μέσα από ένα αριθμό διαφορετικών διαδρομών προς τον τελικό προορισμό και όταν αυτά φθάσουν στον προορισμό τους επανέρχονται στην αρχική τους μορφή . Κάθε συσκευή του δικτύου διαθέτει μια διεύθυνση ip για να διασφαλίσει ότι τα πακέτα θα φθάσουν στον σωστό προορισμό τους. Το βασικό χαρακτηριστικό του ipv6 είναι οι επιπλέον διευθύνσεις στο διαδίκτυο αφού από τα 32 bit του ipv4 μεταπηδήσαμε στα 128 bit με αποτέλεσμα να αυξηθούν οι διευθύνσεις από τις 4 δισεκατομμύρια στις 340 τρισεκατομμύρια .

2.2 Ποιος ο λόγος μετάβασης και χρήσης του ipv6

Ο περιορισμένος αριθμός των διευθύνσεων αποτέλεσε το κυρίαρχο πρόβλημα για την ανάγκη ύπαρξης του ipv6. Ακόμα η απότομη άνθιση των ασύρματων δικτύων είχε ως αποτέλεσμα οι ασύρματες συσκευές να χρειάζονται και αυτές δική τους διεύθυνση οδήγησαν στην ανάγκη της ύπαρξης ενός καλύτερου και πιο αναβαθμισμένου internet protocol .

2.3 Γιατί το ipv6 είναι καλύτερο από το ipv4

Το ipv6 διαθέτει ένα σύνολο από χαρακτηριστικά τα οποία ξεπερνούν τους περιορισμούς του ipv4 . Η πιο προφανής διαφορά είναι οι επιπλέον διευθύνσεις που μπορεί να υποστηρίξει το πρωτόκολλο αυτό, με αποτέλεσμα να μην χρειάζεται πια να χρησιμοποιούμε το πρωτόκολλο NAT το οποίο μας έδινε την δυνατότητα να επεκτείνουμε το ipv4 και έτσι να έχουμε την δυνατότητα να συνδέσουμε πολλές συσκευές σε ένα δίκτυο του τύπου αυτού με την διαφορά όμως να έχουν την ίδια

διεύθυνση με άλλες παγκοσμίως που αυτό είχε σαν αποτέλεσμα να μην είναι ασφαλές , κάτι που το πετυχαίνουμε με το ipv6 αφού μπορούμε να συνδέσουμε πολλές συσκευές και η κάθε μια με την δικιά της ξεχωριστή διεύθυνση . Το ipv6 κάνει ορθή κατανομή των διευθύνσεων με αποτέλεσμα να έχουμε πιο γρήγορη και αποδοτική δρομολόγηση. Που γίνεται λόγω ενός αποδοτικού πίνακα δρομολόγησης ο οποίος υλοποιείται με την χρήση της ιεραρχικής διευθυνσιοδότησης . Η εμβέλεια διεύθυνσης καθορίζει την απαιτούμενη μετάδοση των δεδομένων περιορίζοντας έτσι την περιττή μετάδοση δεδομένων στο δίκτυο. Λόγω της αυτόματης παραμετροποίησης των διευθύνσεων που διευκολύνουν την διαχείριση του δικτύου το πρωτόκολλο αυτό χρησιμοποιείται στις κινητές επικοινωνίες . Η μετάδοση των πακέτων είναι πιο βελτιωμένη στην φωνή και εικόνα παράλληλα με τα δεδομένα με αποτέλεσμα το QoS να είναι καλύτερο .

2.4 Αρχιτεκτονική ipv6

Το ipv6 αυξάνει την ip από 32 bit σε 128 bit . Οι νέες διευθύνσεις απεικονίζονται με την μορφή των οκτώ στοιχείων των 16 bit χωρισμένες με άνω-κάτω τελείες . Για την απεικόνιση των ipv6 διευθύνσεων χρησιμοποιείται η δεκαεξαδική μορφή και όχι η πιο οικία δυαδική μορφή , όπως απεικονίζεται στο παρακάτω παράδειγμα

FE80:0000:0000:0000:0260:97FF:FE8F:64AA

2.5 Ιεραρχική διευθυνσιοδότηση

Η ιεραρχική διευθυνσιοδότηση σχεδιάστηκε για να υποστηρίζει την δρομολόγηση η οποία επιτρέπει την ύπαρξη μικρότερων πινάκων δρομολόγησης και μια περισσότερο αποδοτική κατανομή των διευθύνσεων .Οι μικρότεροι πίνακες δρομολόγησης βελτιώνουν την απόδοση της και παρέχουν γρηγορότερη δρομολόγηση . Μέσω της γρηγορότερης διερεύνησης της διαδρομής και της ελάχιστης καθυστέρησης της απόκρισης.. Με το ipv6 η συσσώρευση επιτυγχάνεται με την προσθήκη ενός προθέματος στην διεύθυνση σε δύο επίπεδα , σε επίπεδο δημόσιας τοπολογίας. Το ipv6 χρησιμοποιεί τις multicast διευθύνσεις αντί για τις broadcast διευθύνσεις .Μια multicast διεύθυνση προσδιορίζει την επικοινωνία οποιοδήποτε πλήθους συγκεκριμένων αποδεκτών, ενώ μια broadcast διεύθυνση προσδιορίζει την επικοινωνία ενός σημείου στο δίκτυο με όλα τα άλλα σημεία σε

αυτό . Το ipv6 προσδιορίζει μία νέου τύπου διεύθυνση την anycast , η οποία καθορίζει την επικοινωνία μεταξύ ενός απλού αποστολέα και τους κοντινότερους αποδέκτες από ένα σύνολο αποδεκτών μιας ομάδας. Όλα τα πακέτα αποστέλλονται σε ένα μέλος της anycast ομάδας , ο αποδέκτης αυτός μπορεί να είναι ένας υπολογιστής ή μια υπηρεσία και επιλέγεται απο το πόσο κοντά βρίσκεται στον αποστολέα με βάση την τοπολογία του . Ένας λόγος που χρησιμοποιείται η anycast δρομολόγηση είναι η αποδοτική ενημέρωση των πινάκων δρομολόγησης .

2.6 Κεφαλίδα πακέτου

Το ipv6 απλοποιεί την πληροφορία της κεφαλίδας έτσι ώστε η επεξεργασία του ip πακέτου να είναι λιγότερο πολύπλοκη . Επιτρέπει τόσο την συμπίεση της κεφαλίδας όσο και τις επιλεκτικές επεκτάσεις αυτής .

Η κεφαλίδα του ipv6 :

- Δεν περιλαμβάνει πλέον το μήκος της κεφαλίδας ,την αναγνώριση τους δείκτες ,την μετατόπιση της κατάτμησης και τον έλεγχο του αθροίσματος στην βασική κεφαλίδα όπου μερικά από αυτά υλοποιούνται αλλού όπως στις κεφαλίδες επέκτασης .

- Είναι σταθερού μεγέθους 40 byte και αμέσως μετά την ipv6 κεφαλίδα συνδέονται οι κεφαλίδες επέκτασης .Χρησιμοποιεί το όριο των αλμάτων αντί του πεδίου του χρόνου ζωής .

- Περιέχει ολοκληρωμένη υποστήριξη του QoS μέσου του πεδίου της κλάσης της κίνησης , το πεδίο αντικαθηστά το πεδίο του τύπου της υπηρεσίας και το πεδίο προσδιορισμού ροής .

2.7 Βελτιώσεις

Το QoS είναι πλήρως ενταγμένο στο ipv6 επιτρέποντας καλύτερη ανάλυση των διαφόρων επιπέδων ποιότητας υπηρεσιών και υποστηρίζοντας την διαφοροποίηση περισσότερων . Στο ipv6 το QoS παρέχεται από την ύπαρξη δύο πεδίων στην κεφαλίδα κάθε ένα από τα οποία ακολουθεί μία συγκεκριμένη μεθοδολογία. Το πρώτο πεδίο είναι το πεδίο κλάσης της κίνησης των 8 bit το οποίο σχεδιάστηκε για να

λειτουργεί με το μοντέλο ποιότητας υπηρεσιών Diffserv, επιτρέποντας έτσι στους κόμβους αποστολή ή στους δρομολογητές μεταβίβασης να καθορίσουν την προτεραιότητα των διαφόρων ipv6 πακέτων, από τα διαφορετικά επίπεδα προτεραιότητας . Το δεύτερο πεδίο είναι το πεδίο προσδιορισμού τς ροής των 20 bit το οποίο σχεδιάστηκε για να λειτουργεί με το IntServ και χρησιμοποιείται για την αίτηση ειδικής μεταχείρισης μιας ακολουθίας ή μιας ροής πακέτων . Η αυθεντικότητα και η κρυπτογράφηση είναι επιβεβλημένη και παρέχεται μέσω του IPsec. Στο ipv6 οι κεφαλίδες επέκτασης της αυθεντικότητας και της κρυπτογράφησης καθορίζονται ξεχωριστά έτσι ώστε οι εφαρμογές υψηλότερου επιπέδου να μπορούν να χρησιμοποιήσουν όταν απαιτείται είτε την μία είτε και τις δύο από τις λειτουργίες αυτές . Επειδή στο ipv6 δεν χρησιμοποιείται το NAT μπορούν με ευκολία να χρησιμοποιήσουν χαρακτηριστικά ασφαλείας . Εν τέλη το ipv6 επειδή είναι μια εξέλιξη του υπάρχοντος ipv4 και όχι μια καινοτομία θα μπει στην ζωή μας με πιο αργούς ρυθμούς από ότι φανταζόμαστε ,δηλαδή μέχρις ότου το υπάρχον πρωτόκολλο να εξαντληθεί . Αναπτύχθηκαν επομένως κάποιες εφαρμογές που χρησιμοποιούμε έτσι ώστε να έχουμε τα οφέλη και των δύο πρωτοκόλλων που είναι οι εξείς :

- **Εφαρμογές διπλής στοίβας(dual stack)**

Διπλή στοίβα ip

Συρράγκωση (tunneling)

Διαμεσολαβητής Σύρραγκας (tunnel Broker)

6-over-4

6-to-4

- **Μετατροπείς διεύθυνσης δικτύου και πρωτοκόλλων**

Stateless IP/ICMP μετατροπέας

Μετατροπή της διεύθυνσης δικτύου –Μετατροπή πρωτοκόλλου (NAT)

Μετατροπέας Μετάδοσης (transport relay translator)

BMP FORMAT

3.1 Τί είναι μία εικόνα BMP

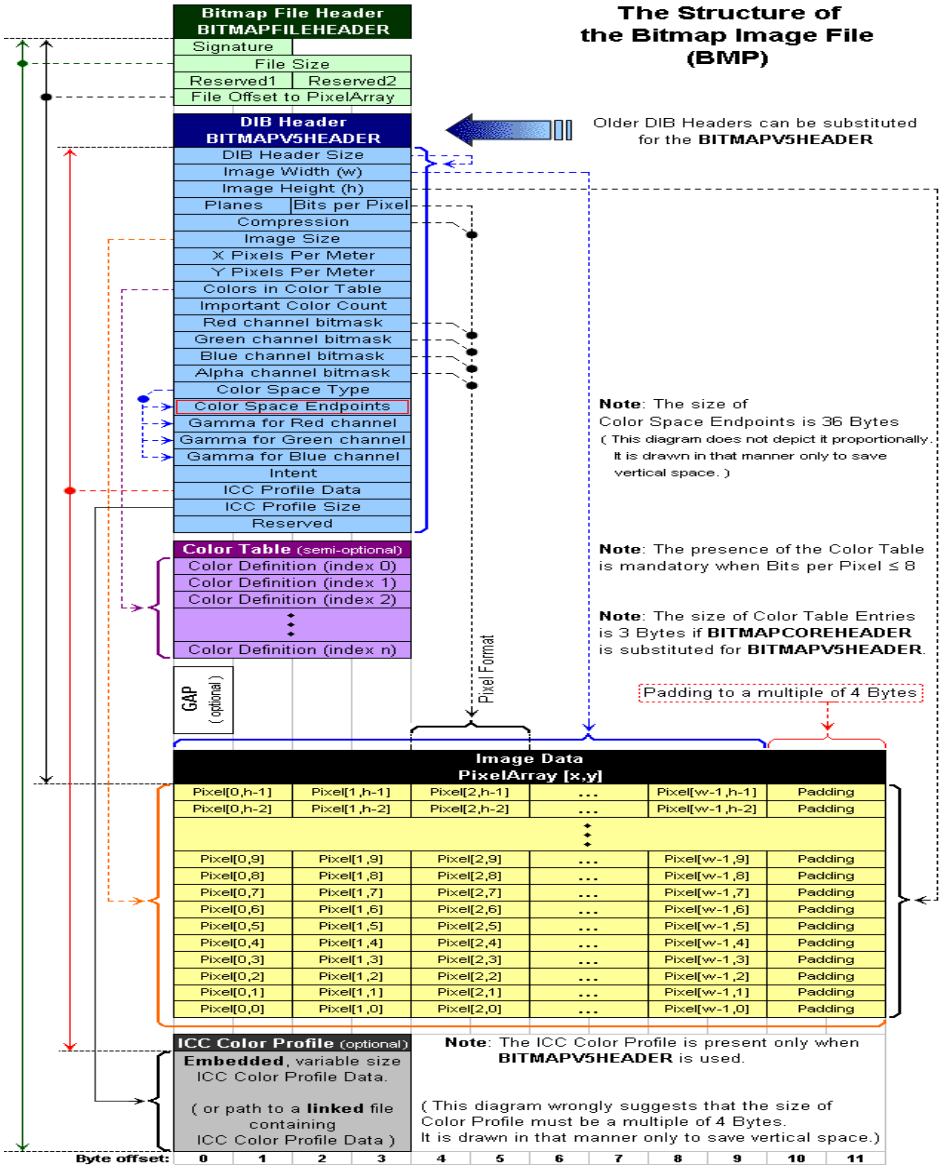
Η μορφή αρχείου BMP, επίσης γνωστή ως αρχείο bitmap ή εικόνα bitmap ανεξάρτητης συσκευής (DIB)μορφής αρχείου ή απλά bitmap,είναι μια εικονογραφημένη μορφή αρχείου raster που χρησιμοποιείται για την αποθήκευση ψηφιακών εικόνων bitmap,ανεξάρτητα από τη συσκευή προβολής όπως μια κάρτα γραφικών. Η μορφή αρχείου BMP είναι σε θέση να αποθηκεύσει 2D ψηφιακές εικόνες ανεξάρτητα το πλάτος και το ύψος,τόσο μονόχρωμης όσο και έγχρωμης εικόνας σε διάφορα βάθη χρωμάτων.

3.2 Εικόνες bitmap ανεξάρτητης συσκευής

Η εικόνα bmp είναι μία δημιουργία της εταιρίας Windows και IBM οι οποίες τρέχουν στην ίδια πλατφόρμα όπου και υποστηρίζουν, για αυτό το λόγο η Windows δημιούργησε τις bitmap εικόνες ανεξάρτητης συσκευής ή αλλιώς DIB-BITMAP. Ως βοήθημα για την ανταλλαγή εικόνων bitmap μεταξύ συσκευών και εφαρμογών κάλεσαν αυτές της μορφής εικόνας ανεξάρτητες από συσκευή ή dibs bitmaps,και η μορφή αρχείου ονομάζεται DIB μορφή αρχείου ή BMP αρχείο εικόνας. Ο κύριος σκοπός της είναι να μετακινηθούν από τη μία συσκευή στην άλλη. Μια DIB είναι μια εξωτερική μορφή, σε αντίθεση με μια συσκευή-εξαρτώμενης bitmap που εμφανίζεται στο σύστημα ως ένα αντικείμενο bitmap που δημιουργήθηκε από μια εφαρμογή των Windows.

3.3 Μορφή αρχείου BMP

Το αρχείο bitmap αποτελείται από σταθερού μεγέθους δομές (HEADERS) καθώς και με μεταβλητό μέγεθος δομής που εμφανίζονται σε μια προκαθορισμένη ακολουθία. Πολλές διαφορετικές εκδόσεις ορισμένων από αυτές τις δομές μπορεί να εμφανιστούν στο αρχείο, λόγω της μακράς εξέλιξης αυτής της μορφής αρχείου.



Σχήμα 4. Δομή Αρχείου BMP

Όπως βλέπουνε από την εικόνα η δομή μιας bmp εικόνας αρχικά αποτελείται από την κεφαλίδα αρχείου η οποία έχει μέγεθος 14 bytes, σκοπός της είναι να αρχειοθετήσει τις πληροφορίες σχετικά με το αρχείο εικόνας bmp και χρησιμοποιείται μέχρις ότου το αρχείο φορτωθεί στην μνήμη. Μετά ακολουθεί η κεφαλίδα ανεξάρτητης συσκευής bitmap όπου υπάρχουν 7 βελτιωμένες εκδώσεις και χρησιμοποιείται για να αποθηκεύει λεπτομερείς πληροφορίες σχετικά με την bitmap εικόνα και καθορίζει την μορφοποίηση των εικονοστοιχείων. Αμέσως μετά ακολουθούν επιπλέον bits με πληροφορίες σχετικά με την εικόνα όπως το μήκος το πλάτος και τα χρώματα στον πίνακα χρωμάτων που θα χρησιμοποιηθούν αργότερα. Ουσιαστικά χρησιμοποιείται για την σωστή μορφοποίηση των εικονοστοιχείων και

έχει μέγεθος 12 ή 16 bytes και εκτελείται υπό τις προϋποθέσεις μόνο όταν η κεφαλίδα DIB είναι η βασική πληροφοριοδότης για την μορφοποίηση της εικόνας . Μετά ακολουθεί ο πίνακας χρωμάτων που έχει μεταβλητό μέγεθος και καθορίζει τα χρώματα που θα χρησιμοποιηθούν για τον πίνακα των εικονοστοιχείων της bitmap εικόνας με βάθος χρώματος της εικόνας μέχρι 8. Η άμεση ένωση του πίνακα χρωμάτων με το πίνακα εικονοστοιχείων εικόνας bitmap λέγεται Gap1 το οποίο έχει μεταβλητό μέγεθος και βασικός του σκοπός είναι για την ευθυγράμμιση της δομής του αρχείου. Ο πίνακας εικονοστοιχείων έχει μεταβλητό μέγεθος και καθορίζει τις ακριβείς τιμές των εικονοστοιχείων. Η μορφοποίηση των εικονοστοιχείων καθορίζεται από την κεφαλίδα DIB και τον πίνακα επιπλέον bits. Κάθε γραμμή του πίνακα των εικονοστοιχείων γεμίζεται με πολλαπλάσια των 4 bytes. Όπως το Gap1 έτσι και το Gap2 χρησιμοποιείται για την ευθυγράμμιση της δομής του αρχείου έχει μεταβλητό μέγεθος .Το τελευταίο κομμάτι είναι το προφίλ χρωμάτων ICC το οποίο έχει μεταβλητό μέγεθος και χρησιμοποιείται για την διαχείριση του προφίλ χρωμάτων. Μπορεί επίσης να περιέχει μια διαδρομή προς ένα εξωτερικό αρχείο που περιέχει το προφίλ χρωμάτων. Όταν φορτώνονται ως μη συσκευασμένα DIB,τοποθετείται μεταξύ του πίνακα χρωμάτων και gap1.

3.4 Η λειτουργία της DIB-BITMAP εικόνας στην μνήμη

Ένα αρχείο εικόνας bitmap αφού φορτώνεται στη μνήμη γίνεται ανεξάρτητη δομή δεδομένων .Η ανεξάρτητη δομή δεδομένων μιας bitmap εικόνας είναι ίδια με την μορφοποίηση του bitmap αρχείου. Αλλά δεν περιέχει την κεφαλίδα αρχείου bitmap των 14 bytes και ξεκινά με την κεφαλίδα DIB. Για τις κεφαλιδες DIB που φορτώνονται στη μνήμη ο πίνακας χρωμάτων μπορεί να περιέχει επίσης 16 καταχωρίσεις bit, όπου αποτελούν δείκτες για την τρέχουσα κατάσταση. Σε όλες τις περιπτώσεις ο πίνακας εικονοστοιχείων πρέπει να ξεκινά με μια διεύθυνση μνήμης πολλαπλάσια των 4 bytes. Σε μι συσκευασμένες DIB εικόνες που φορτώνονται στη μνήμη τα επιλεγμένα δεδομένα του προφίλ χρωμάτων θα πρέπει να τοποθετούνται αμέσως μετά τον πίνακα χρωμάτων και πριν τον πίνακα εικονοστοιχείων και του Gap1. Όταν το μέγεθος του gap1 και gap2 είναι μηδέν, η DIB δομή των δεδομένων αναφέρεται ως συσκευασμένα DIB και μπορεί να αναφέρεται από ένα ενιαίο δείκτη δείχνοντας την αρχή της κεφαλίδας DIB. Σε όλες τις περιπτώσεις, η διάταξη των εικονοστοιχείων αρχίζουν σε μια διεύθυνση μνήμης που είναι πολλαπλάσια των 4

byte. Σε ορισμένες περιπτώσεις μπορεί να χρειαστεί να ρυθμίσετε τον αριθμό των καταχωρήσεων στον πίνακα χρωμάτων προκειμένου να αναγκάσει τη διεύθυνση μνήμης του πίνακα εικονοστοιχείων σε πολλαπλάσιο των 4 bytes.

3.5 Κεφαλίδα αρχείου bitmap

Αυτό το μπλοκ των bytes που βρίσκεται στην αρχή του αρχείου και χρησιμοποιείται για να προσδιορίσει το αρχείο. Μια τυπική εφαρμογή διαβάζει αυτό το μπλοκ πρώτον για να εξασφαλίσει ότι το αρχείο είναι στην πραγματικότητα ένα αρχείο BMP καθότι δεν έχει φθαρεί. Τα πρώτα δύο bytes της μορφή αρχείου BMP είναι του χαρακτήρα «B», Τότε ο χαρακτήρας «M»στο 1-byte κωδικοποίηση ASCII. Όλες οι ακέραιες τιμές αποθηκεύονται σε little-endian μορφή (δηλαδή λιγότερο σημαντικό byte πρώτο).

3.6 DIB κεφαλίδα (κεφαλίδα bitmap πληροφορίες)

Αυτό το μπλοκ των bytes που αφηγείται την εφαρμογή σχετικά με την εικόνα,το οποίο θα χρησιμοποιηθεί για την εμφάνιση της εικόνας στην οθόνη. Το μπλοκ συμφωνεί επίσης με την κεφαλίδα που χρησιμοποιείται εσωτερικά από τα Windows και έχει πολλές παραλλαγές. Όλα αυτά περιέχουν ένα DWORD(32 bit) προσδιορίζοντας το μέγεθός τους, πράγμα που σημαίνει ότι μπορούν εύκολα να προσδιορίσουν ποια κεφαλίδα χρησιμοποιείται στην εικόνα. Ο λόγος ότι υπάρχουν διαφορετικές κεφαλίδες είναι ότι η Microsoft επεκτείνεται με τη μορφή DIB αρκετές φορές. Οι νέες διευρυμένες κεφαλίδες μπορεί να χρησιμοποιηθούν με ορισμένες λειτουργίες του GDI αντί για τους παλαιότερους, παρέχοντας περισσότερη λειτουργικότητα. Δεδομένου ότι το GDI υποστηρίζει μια λειτουργία για τη φόρτωση αρχείων bitmap, τυπικές εφαρμογές των Windows χρησιμοποιούν τη λειτουργικότητα. Μία συνέπεια αυτού είναι ότι για τέτοιες εφαρμογές, οι μορφές BMP που υποστηρίζουν ταιριάζουν με τις μορφές που υποστηρίζονται από την έκδοση των Windows που τρέχει.

3.7 Πίνακας χρωμάτων

Ο πίνακας χρωμάτων (παλέτα) εμφανίζεται στο αρχείο εικόνας BMP αμέσως μετά την κεφαλίδα του αρχείου BMP, η επικεφαλίδα DIB (προαιρετικά με τρία χρώματα κόκκινο, πράσινο και μπλε bitmasks αν η επικεφαλίδα του BITMAPINFOHEADER

με επιλογή BI_BITFIELDS χρησιμοποιείται). Ως εκ τούτου, αντισταθμίζεται το μέγεθος της BITMAPFILEHEADER συν το μέγεθος της επικεφαλίδας DIB(συν 12 bytes προαιρετικό για τις τρεις μάσκες bit). Ο αριθμός των καταχωρήσεων στην παλέτα είναι είτε 2^n ή ένας μικρότερος αριθμός που καθορίζεται στην κεφαλίδα (στο OS / 2 μορφοποίησης επικεφαλίδας του BITMAPCOREHEADER , μόνο το πλήρες μέγεθος παλέτας υποστηρίζεται). Στις περισσότερες περιπτώσεις, κάθε καταχώρηση στον πίνακα καταλαμβάνει 4 byte, με τη σειρά μπλε, πράσινο,κόκκινο,0x00. Ο πίνακας χρωμάτων είναι ένα μπλοκ από bytes . Κάθε εικονοστοιχείο σε μια εικόνα περιγράφεται από έναν αριθμό bits (1, 4, ή 8) η οποία αποτελεί δείκτη του ένα μόνο χρώμα που περιγράφεται από αυτόν τον πίνακα. Ο σκοπός της παλέτας χρωμάτων στο ευρετήριο bitmaps είναι να ενημερώσει την εφαρμογή για το πραγματικό χρώμα όπου κάθε μία από αυτές τις τιμές του δείκτη αντιστοιχεί. Ο σκοπός του πίνακα χρωμάτων σε μη-δείκτη bitmaps είναι στη λίστα με τα χρώματα που χρησιμοποιούνται από το bitmap για τους σκοπούς της βελτιστοποίησης για συσκευές με περιορισμένες δυνατότητες να διευκολύνουν τη μελλοντική μετατροπή σε διαφορετικές μορφές εικονοστηχείων. Τα χρώματα στον πίνακα χρωμάτων που συνήθως ορίζονται στα 4-byte ανά εγγραφή 8.8.8.0.8 (σε σημειογραφία RGBAX). Ο πίνακας που χρησιμοποιείται στα χρώματα του BITMAPCOREHEADER με το OS / 2 χρησιμοποιεί 3-byte ανά εγγραφή 8.8.8.0.0 .Οι DIB εικόνες φορτώνονται στη μνήμη, ο πίνακας χρωμάτων μπορεί προαιρετικά να αποτελείται από 2-byte .. Αυτές οι καταχωρήσεις αποτελούν δείκτες για την τρέχουσα παλέτα . Η Microsoft δεν απαγορεύει την παρουσία ενός έγκυρου καναλιού με μάσκα bit και BITMAPV4HEADER BITMAPV5HEADER για 1bpp, 4bpp και 8bpp έγχρωμων εικόνων, πράγμα που δείχνει ότι η καταχώριση του πίνακα χρωμάτων μπορεί επίσης να καθορίσει ένα στοιχείο άλφα χρησιμοποιώντας το 8.8.8. [0-8]. [0-8] μορφή μέσω του RGBQUAD.rgbReserved . Ωστόσο, ορισμένες εκδόσεις που απαγορεύει την τεκμηρίωση τους η Microsoft, δηλώνοντας ότι η RGBQUAD.rgbReserved μέλος "πρέπει να είναι μηδέν. Όπως προαναφέρθηκε, ο πίνακας χρωμάτων συνήθως δεν χρησιμοποιείται όταν τα εικονοστοιχεία είναι 16-bit ανά εικονοστοιχείο (16bpp) . Κανονικά δεν υπάρχουν καταχωρήσεις πίνακα χρωμάτων σε αυτά τα αρχεία bitmap .Ωστόσο, η τεκμηρίωση της Microsoft διευκρινίζει ότι για 16bpp και υψηλότερα, ο πίνακας χρωμάτων μπορεί να αποθηκεύετε σε μια λίστα με τα χρώματα που προορίζονται για τη βελτιστοποίηση με περιορισμένη δυνατότητα έγχρωμης οθόνης, ενώ καθορίζει επίσης, ότι σε τέτοιες περιπτώσεις, δεν αναπροσαρμόζονται

καταχωρήσεις παλέτα σε αυτόν τον πίνακα χρωμάτων. Αυτό μπορεί να φαίνεται σαν μια αντίφαση, εάν δεν γίνεται διάκριση μεταξύ των υποχρεωτικών καταχωρήσεων παλέτα και την προαιρετική λίστα χρωμάτων.

3.8 Αρχειοθέτηση εικονοστοιχείων

Τα bits που αντιπροσωπεύουν τα εικονοστοιχεία bitmap είναι συσκευασμένα σε σειρές. Το μέγεθος της κάθε σειράς στρογγυλοποιείται στο ανώτερο πολλαπλάσιο των 4byte (32bit DWORD).Για εικόνες με ύψος>1,οι πολλαπλές σειρές αποθηκεύονται διαδοχικά, σχηματίζοντας ένα πίνακα εικονοστοιχείων. Ο συνολικός αριθμός των bytes που απαιτούνται για να αποθηκεύσετε μία σειρά από pixels μπορεί να υπολογιστεί ως εξής:

$$\mathrm{RowSize} = \left\lceil \frac{\mathrm{BitsPerPixel} \cdot \mathrm{ImageWidth}}{32} \right\rceil \cdot 4,$$

Το συνολικό ποσό των bytes που απαιτούνται για να αποθηκεύσετε μια σειρά από εικονοστοιχεία σε n bits ανά εικονοστοιχείο (BPP), με 2^n χρώματα, μπορεί να υπολογιστεί από συνυπολογισμό της επίδρασης της στρογγυλοποίησης στο μέγεθος της κάθε γραμμής σε πολλαπλάσιο των 4 byte, ως εξής:

$$\mathrm{PixelArraySize} = \mathrm{RowSize} \cdot |\mathrm{ImageHeight}|$$

Το ImageHeight εκφράζεται σε εικονοστοιχεία. Η απόλυτη τιμή είναι αναγκαία, διότι το ImageHeight μπορεί να έχει αρνητική τιμή .Η συνολική εικόνα bitmap του αρχείου μπορεί να προσεγγιστεί ως:

$$\mathrm{FileSize} \approx 54 + 4 \cdot 2^{\mathrm{bpp}} + \mathrm{PixelArraySize}$$

Για BPP≤8 (επειδή για εικονοστοιχεία μεγαλύτερα από 8 bit, η παλέτα δεν είναι υποχρεωτική).

Μόνο με εικόνες 8 ή λιγότερα bit ανά εικονοστηχεία πρέπει να αντιπροσωπεύουν την παλέτα. 16Bp εικόνες (ή μεγαλύτερο), μπορεί να παραλείψει το μέρος της παλέτας από τον υπολογισμό του μεγέθους, ως εξής:

$$\mathrm{FileSize} \approx 54 + \mathrm{PixelArraySize}$$

Για bits ανά εικονοστοιχείο >8 ο αριθμός 54 είναι το συνολικό μέγεθος του bitmap με κεφαλίδα αρχείου 14 byte και τα 40 byte η δημοφιλής κεφαλίδα WindowsDIB η έκφραση 4*2^n είναι το μέγεθος της παλέτας χρωμάτων σε bytes.Αυτό το συνολικό μέγεθος του αρχείου είναι μόνο μια προσέγγιση,δεδομένου ότι το μέγεθος της παλέτας χρωμάτων θα είναι 3*2^n byte για το OS/2 έκδοση του BITMAPCOREHEADER με κεφαλίδα DIB, και κάποια αρχεία μπορεί να καθορίζουν μόνο τον αριθμό των χρωμάτων που απαιτούνται συγκεκριμένα λιγότερα από 2^n . Με πρόσθετη αβεβαιότητα το μέγεθος εισάγεται από την προαιρετική παρουσία για τις επιπλέον μάσκες αμέσως μετά την επικεφαλίδα BITMAPINFOHEADER DIB, και το μεταβλητό μέγεθος GAP απεικονίζεται στο Diag.1

3.9 Πίνακας εικονοστοιχείων

Ο πίνακας εικονοστοιχείων είναι μια διάταξη μπλοκ των 32-bit DWORDs, που περιγράφει την εικόνα εικονοστοιχείο προς εικονοστοιχείο .Κανονικά τα εικονοστοιχεία αποθηκεύονται ανάποδα σε σχέση με την κανονική εικόνα , ξεκινά από την αριστερή γωνία και πηγαίνει από τα αριστερά προς τα δεξιά και μετά στην επόμενη γραμμή από κάτω προς τα πάνω κατά μήκος της εικόνας . Αποσυμπιεσμένες εικόνες windows bitmap μπορούν να αρχειοθετηθούν από την κορυφή προς τα κάτω όταν η τιμή του ύψους είναι αρνητική. Στην αυθεντική έκδοση της κεφαλίδας του OS/2 οι νόμιμες τιμές του βάθους του χρώματος είναι 1,4,8 και 24 bit ανά εικονοστοιχείο (BPP). Η συμπλήρωση ψηφίων πρέπει να γίνεται στο τέλος των γραμμών προκειμένου να θέσει το μήκος της γραμμής σε πολλαπλάσιο των 4 bytes. Όταν ο πίνακας εικονοστοιχείων φορτώνεται στη μνήμη κάθε γραμμή πρέπει να αρχίσει σε μια διεύθυνση μνήμης πολλαπλάσια του 4. Αυτός ο περιορισμός είναι μόνο για τον πίνακα εικονοστοιχείων που έχει φορτωθεί στη μνήμη . Για σκοπούς αρχειοθέτησης μόνο το μέγεθος της γραμμής πρέπει να είναι πολλαπλάσιο των 4 bytes. Μια εικόνα bitmap 24 bit με μήκος ίσον με 1 θα έπρεπε να έχει 3 bytes ανά γραμμή και 1 byte για να γεμίσει , αν έχει μήκος ίσον με 2 τότε θα έχει 2 byte για να

γεμίσει αν έχει μήκος ίσον με 3 θα έχει 3 byte για να γεμίσει ενώ αν έχει μήκος ίσον με 4 δεν θα έχει πλέον γέμισμα.

3.10 Συμπίεση

Οι έγχρωμες εικόνες μπορούν να συμπιεστούν με 4 bit ή 8 bit βάση του αλγορίθμου RLE η του HUFFMAN 1D . Οι εικόνες 24 bit OS/2 BITMAPCOREHEADER μπορούν να συμπιεστούν με 24 BPP με τον αλγόριθμο RLE. Οι εικόνες με 16 και 32 BPP μπορούν να αρχειοθετηθούν αποσυμπιεστές . Όλες οι εικόνες κάθε βάθους όλων των χρωμάτων μπορούν να αρχειοθετηθούν αποσυμπιεστές.

3.11 Μορφοποίηση εικονοστοιχείων

Το 1bit ανά εικονοστοιχείο υποστηρίζει 2 διαφορετικά χρώματα . Οι τιμές των εικονοστοιχείων αποθηκεύονται σε κάθε κομμάτι, με το πρώτο (από τα αριστερά) εικονοστοιχείο στο πιο σημαντικό του πρώτου byte. Κάθε κομμάτι είναι ένας δείκτης σε ένα πίνακα 2 χρωμάτων. Αυτός ο πίνακας χρωμάτων είναι σε μορφή 32bpp 8.8.8.0.8.

Τα 2bit ανά pixel (2bpp) υποστηρίζει 4 διαφορετικά χρώματα και 4 εικονοστοιχεία ανά 1 byte, το πιο αριστερά-εικονοστοιχείο είναι στα δύο πιο σημαντικά κομμάτια (μόνο στα Windows CE). Κάθε τιμή είναι ένας 2-bit δείκτης σε ένα πίνακα μέχρι 4 χρώματα .Αυτός ο πίνακας χρωμάτων είναι σε μορφή 32bpp 8.8.8.0.8 RGBAX.

Τα 4-bit ανά εικονοστοιχείο (4bpp) μορφή υποστηρίζει 16 διαφορετικά χρώματα και 2 εικονοστοιχεία ανά 1 byte, το πιο αριστερά εικονοστοιχείο είναι το πιο σημαντικό. Κάθε τιμή εικονοστοιχείου είναι ένας 4-bit δείκτης σε ένα πίνακα έως και 16χρώματα. Αυτός ο πίνακας χρωμάτων είναι σε μορφή 32bpp 8.8.8.0.8 RGBAX.

Τα 8bit ανά εικονοστοιχείο (8bpp) υποστηρίζει τη μορφή 256 διαφορετικών χρωμάτων και 1 pixel ανά 1 byte. Κάθε byte είναι ένας δείκτης σε ένα πίνακα έως 256 χρώματα. Αυτός ο πίνακας χρωμάτων είναι σε μορφή 32bpp 8.8.8.0.8RGBAX.

Τα 16bit ανά εικονοστοιχείο (16bpp) υποστηρίζει τη μορφή διαφορετικών χρωμάτων 65536και 1 εικονοστοιχείο ανά 2 byte Κάθε λέξη μπορεί να καθορίσει το Άλφα

κόκκινο, πράσινο και μπλε δείγμα του εικονοστοιχείου .

Τα 24bit ανά εικονοστοιχείο (24bpp) υποστηρίζει 16.777.216 χρώματα αξίας 1 εικονοστοιχείο ανά 3 byte. Κάθε τιμή εικονοστοιχείων ορίζει το κόκκινο,πράσινο και μπλε δείγμα του εικονοστοιχείων (8.8.8.0.0 σε σημειογραφία RGBAX). Συγκεκριμένα με τη σειρά (μπλε, πράσινο και κόκκινο, 8-bit ανά κάθε δείγμα).

Τα 32bit ανά εικονοστηχείο (32bpp) υποστηρίζει 4.294.967.296 διαφορετικά χρώματα με 1 εικονοστοιχείο ανά 4 byte DWORD. Κάθε DWORD μπορεί να καθορίσει το Άλφα, κόκκινο, πράσινο και μπλε δείγμα του εικονοστοιχείου

ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ

4.1 Περιγραφή του τι θα αναλύσουμε

Όπως αναλύσαμε και στην περιληψη ο σκοπός του βιβλίου είναι να μας περιγράψει την διαδικασία αποστολής μιάς εικόνας απο έναν αποστολέα σε έναν δέκτη, αρχικά αναλύουμε το είδος του δικτύου στο οποιίο μεταδήδουμε την εικόνα ,όπου στην περίπτωση μας είναι το multicasting όπου και το αναλύουμε διεξωδικά στο πρώτο κεφάλαιο . Μετά αναλύουμε το είδος του πρωτοκόλλου δικτύου που μεταδίδουμε την εικόνα όπου στην περίπτωση μας είναι το internet protocol version 6(ipv6), όπου το αναλύουμε στο δεύτερο κεφάλαιο . Αμέσως μετά αναλύουμε το είδος την εικόνας που στέλνουμε μέσω του δικτύου , στην περίπτωση μας η διαμόρφωση της εικόνας ονομάζεται BMP την οποίοα διαμόρφωση αναλύουμε στο τρίτο κεφάλαιο.

Σκοπός μου δεν είναι μόνο να σας περιγράψω γραμμή γραμμή τον κώδικα αλλά να σας περιγράψω μια πιό σαφή εικόνα των στοιχείων που παίζουν ρόλο για να φτάσεις να γράψεις τον κώδικα ,έτσι ώστε ακόμα και ένας αρχάριος απο υπολογιστές όταν ανοίξει το βιβλίο και τυχαία ή διαβάζοντας το φτάσει στο τέταρτο κεφάλαιο να ξέρει τι πρέπει να κάνει για να γράψει τον αντόιστιχο κώδικα .

Αρχικά θα ξεκινήσουμε με το γεγονός ότι ο κώδικας τρέχει σε μια πλατφόρμα που ονομάζεται LINUX. Στην περιγραφή που θα ακολουθήσει θα αναλύσουμε τι είναι το εξής λειτουργικό και πώς θα εγκαταστήσουμε τους μεταφραστές/compilers για να τρέξουμε τον κώδικα . Ακόμα θα σας περιγράψουμε όπως είναι λογικό την γλώσσα στην οποία γράψαμε τον κώδικα και τι εντολές πρέπει να γράψουμε στο τερμάτικό για να τρέξουμε τον κώδικα για να δούμε το αποτέλεσμα που επιθυμούμε.

4.2 Ανάλυση για το λειτουργικό που τρέχουμε και τι είναι η γλώσσα προγραμματισμού C

Ας ξεκινήσουμε λοιπόν με το λειτουργικό στο οποίο τρέχουμε τον κώδικα . Το λειτουργικό ονομάζεται LINUX.

Το LINUX είναι ένας γενικός όρος αναφοράς σε λειτουργικά συστήματα που βασίζονται στον πυρήνα Linux. Η αρχιτεκτονική του Linux είναι βασισμένη στις αρχές του λειτουργικού Unix αλλά έχει αναπτυχθεί εκ του μηδενός και δεν περιλαμβάνει κώδικα από το Unix. Η ανάπτυξη του Linux είναι χαρακτηριστικό παράδειγμα εθελοντικής συνεργασίας από διαδικτυακές κοινότητες, ενώ όλο το έργο είναι ανοικτού κώδικα και ελεύθερα προσβάσιμο από όλους για αντιγραφή, τροποποίηση ή αναδιανομή χωρίς περιορισμό. Το Linux είναι διαθέσιμο υπό άδειες όπως η GNU General Public License. Το Linux μπορεί να εγκατασταθεί και να λειτουργήσει σε μεγάλη ποικιλία υπολογιστικών συστημάτων, από μικρές συσκευές όπως κινητά τηλέφωνα μέχρι μεγάλα υπολογιστικά συστήματα και υπερυπολογιστές. Χρησιμοποιείται κατά κόρον σε διακομιστές, αφού η καταγεγραμμένη χρήση Linux σε διακομιστές για το 2008 ανέρχεται σε 60% του συνόλου της αγοράς. Οι περισσότεροι προσωπικοί υπολογιστές όμως, λειτουργούν με Mac OS X ή Microsoft Windows, καθώς το αντίστοιχο ποσοστό του Linux είναι μόλις 5%. Τα τελευταία χρόνια πάντως παρατηρείται άνοδος του Linux και σε προσωπικούς υπολογιστές, χάρη στη δημοφιλή διανομή Ubuntu αλλά και τις περισσότερες λειτουργίες που προσφέρει σε συστήματα με περιορισμένες δυνατότητες όπως τα netbook. Το Linux κυκλοφορεί σε *διανομές Linux*, δηλαδή ο πυρήνας σε συνδυασμό με συνοδευτικά προγράμματα, όπως βιβλιοθήκες, εργαλεία συστήματος, παραθυρικό περιβάλλον εργασίας και πολλές άλλες εφαρμογές που απαιτούνται για την εύρυθμη λειτουργία ενός υπολογιστή. Σχεδόν όλες οι διανομές περιλαμβάνουν το πρόγραμμα περιήγησης Mozilla Firefox και τη σουίτα εφαρμογών γραφείου Libreoffice. Χαρακτηριστικό των διανομών είναι η μεγάλη δυνατότητα παραμετροποίησης και επιλογής που προσφέρουν καθώς κάθε μια απευθύνεται σε διαφορετικό τύπο χρηστών. Ανάλογα με την φιλοσοφία που ακολουθεί κάθε διανομή μπορεί να δίνει μεγαλύτερη βάση στη

φιλικότητα προς τον χρήστη, στις εφαρμογές πολυμέσων, την ευκολία αραμετροποίησης κ.α.Το Linux έχει ως πρότυπο ανάπτυξης το ίδιο το Unix. Το λειτουργικό σύστημα Unix σχεδιάστηκε και αναπτύχθηκε το 1969, από τους Ken Thompson, Dennis Ritchie, Douglas McIlroy και Joe Ossanna, για λογαριασμό της αμερικανικής εταιρείας AT&T. Κυκλοφόρησε για πρώτη φορά το 1971 και ήταν εξολοκλήρου γραμμένο σε συμβολική γλώσσα (assembly), κάτι που ήταν σύνηθες εκείνη την εποχή. Δύο χρόνια αργότερα, κυκλοφόρησε μια νέα, πρωτοποριακή έκδοση του Unix γραμμένη σε γλώσσα C από τον Dennis Ritchie. Επειδή πλέον βασιζόταν σε μια γλώσσα υψηλού επιπέδου έγινε πιο εύκολη η μεταφορά του σε περισσότερες πλατφόρμες υπολογιστών, καθιστώντας το Unix πολύ δημοφιλές σε ακαδημαϊκά ιδρύματα και επιχειρήσεις.

Το εγχείρημα GNU, που εκπόνησε ο Ρίτσαρντ Στώλλμαν το 1983, έχει ως στόχο την δημιουργία ενός ολοκληρωμένου πακέτου με ελεύθερο λογισμικό συμβατό με το Unix. Η ανάπτυξη του GNU ξεκίνησε το 1984, ενώ ο Stallman ίδρυσε το Ίδρυμα Ελεύθερου Λογισμικού το 1985 και το 1989 εξέδωσε την άδεια χρήσης GNU General Public License (GNU GPL). Στις αρχές του '90, είχε ολοκληρωθεί η ανάπτυξη χρήσιμων εργαλείων που απαιτούνται από ένα λειτουργικό (όπως βιβλιοθήκες, μεταγλωττιστές, επεξεργαστές κειμένου, κέλυφος, παραθυρικό περιβάλλον), αλλά είχε καθυστερήσει η ανάπτυξη βασικών και αναγκαίων εργαλείων όπως οι οδηγοί υλικού, οι δαίμονες εργασιών αλλά και ο πυρήνας του λειτουργικού. Αυτή η καθυστέρηση εξώθησε τον Φινλανδό Linus Torvalds να δημιουργήσει τον δικό του πυρήνα το 1991.

Ο Torvalds ξεκίνησε την ανάπτυξη ενός μη-εμπορικού unix-οειδές λειτουργικού το 1991, ενώ φοιτούσε ακόμα στο Πανεπιστήμιο του Ελσίνκι. Επηρεάστηκε από το επίσης unix-οειδές λειτουργικό MINIX, και άρχισε να αναπτύσσει αυτό που αργότερα έγινε γνωστό ως πυρήνας Linux. Το MINIX, είναι ένα μινιμαλιστικό λειτουργικό παρόμοιο με το Unix, που αναπτύχθηκε από τον Άντριου Τανενμπάουμ για εκπαιδευτικούς σκοπούς. Ο Torvalds αρχικά έγραφε προγράμματα που έτρεχαν και στο MINIX έως ότου το Linux έφτασε σε ένα στάδιο ανάπτυξης όπου δεν ήταν πλέον απαραίτητοι οι δεσμοί μεταξύ των δυο λειτουργικών. Έπειτα, ο Tovalds αποφάσισε να αλλάξει την άδεια χρήσης, που μέχρι τότε δεν επέτρεπε την αναδιανομή για εμπορικούς σκοπούς, κάνοντας διαθέσιμο το Linux υπό την άδεια GNU GPL. Έτσι το GNU βρήκε έναν πυρήνα για να λειτουργήσει, και το Linux

βρήκε έτοιμη μια μεγάλη ποικιλία προγραμμάτων. Εντάσσοντας το εγχείρημά του στο GNU, η ανάπτυξη του Linux ήταν αλματώδης και γρήγορα ξεπέρασε το MINIX.
Από την προσχώρηση του Linux στο GNU μέχρι σήμερα, χιλιάδες προγραμματιστές από όλο τον κόσμο συνεισφέρουν κώδικα και αναπτύσσουν από κοινού το Linux. Κάθε διανομή υποστηρίζεται από μια οργανωμένη κοινότητα χρηστών και προγραμματιστών, ενώ ορισμένες από τις διανομές υποστηρίζονται και από εταιρίες που πωλούν είτε εμπορικές εκδόσεις είτε τεχνική υποστήριξη για δωρεάν εκδόσεις. Επιπλέον, δεκάδες τρίτες εταιρίες έχουν συνεισφέρει τα τελευταία χρόνια στην ανάπτυξη του Linux - ανάμεσα στις οποίες πολύ γνωστές όπως η IBM, η Intel, η Google, η Hewlett Pacard - κυρίως για να αυξήσουν τις πωλήσεις harware τους - με δεδομένη τη διάδοση του Linux στην αγορά των διακομιστών, των κινητών τηλεφώνων και των netbooks. Το Linux αναπτύσσεται με βάση το πρότυπο POSIX, το οποίο είναι μία προσπάθεια τυποποίησης όλων των συστημάτων που βασίζονται ή προσομοιώνουν το UNIX.
Η κύρια διαφορά μεταξύ του Linux και άλλων δημοφιλών λειτουργικών είναι ότι ο πυρήνας του Linux άλλα και οι σημαντικότερες εφαρμογές του είναι ελεύθερο και ανοικτού κώδικα λογισμικό. Υπάρχουν και άλλα λειτουργικά που κυκλοφορούν με την ίδια άδεια, αλλά το Linux είναι το πιο διαδεδομένο. Ορισμένες άδειες για ελεύθερο και ανοικτού κώδικα λογισμικό βασίζονται στην αρχή του copyleft, ένα είδος αμοιβαιότητας όπου κάθε νέο εγχείρημα που παράγεται από ένα έργο copyleft πρέπει να είναι επίσης copyleft. Η πιο δημοφιλής άδεια ελεύθερου λογισμικού, η GNU GPL, είναι μια μορφή copyleft, και χρησιμοποιείται για τον πυρήνα Linux και για αρκετές εφαρμογές του εγχειρήματος GNU.

Αφού αναλύσαμε αρχικά ποιο είναι το λινουξ ,τώρα θα σας περιγράψω την έκδοση λινουξ που έτρεξα στον υπολογιστή μου η οποία είναι η έκδοση ubuntu λινουξ.

Το **Ubuntu** είναι ένα ανοικτού κώδικα, ελεύθερο και δωρεάν λειτουργικό σύστημα βασισμένο στον πυρήνα Linux. Το όνομά του προέρχεται από την έννοια *ubuntu* των Ζουλού και Κόσα (Xhosa), που σημαίνει "Είμαι ότι είμαι λόγω όσων όλοι είμαστε". Το Ubuntu ξεκίνησε το 2004, βασισμένο στη διανομή Debian. Ο στόχος του Ubuntu είναι η παροχή ενός διαρκώς ενημερωμένου, σταθερού λειτουργικού συστήματος για τον μέσο χρήστη, με ενισχυμένη έμφαση στην ευκολία χρήσης και εγκατάστασης. Το Ubuntu έχει χαρακτηριστεί ως η πιο δημοφιλής

διανομή Linux για επιτραπέζιους υπολογιστές, διεκδικώντας περίπου το 30% επί του συνόλου των Linux συστημάτων σύμφωνα με έρευνα του 2007. Το Ubuntu είναι ελεύθερο και ανοικτού κώδικα λειτουργικό, που σημαίνει ότι διανέμεται χωρίς χρέωση αλλά και ότι μπορεί να βελτιωθεί από κάθε προγραμματιστή που θέλει να συμμετάσχει στην ομάδα ανάπτυξης. Το Ubuntu χρηματοδοτείται από την Canonical Ltd., μία ιδιωτική επιχείρηση που ιδρύθηκε από τον Νοτιοαφρικανό επιχειρηματία Μαρκ Σάτλγουορθ. Αντί να πωλεί το Ubuntu καθεαυτό, η Canonical καταγράφει έσοδα από την επί πληρωμή τεχνική υποστήριξη που παρέχει για το προϊόν της. Διατηρώντας το Ubuntu ελεύθερο και ανοικτό η Canonical δέχεται και την βοήθεια τρίτων προγραμματιστών για την ανάπτυξή του. Χρησιμοποιεί επίσης εφαρμογές και κώδικα της διανομής Debian, από την οποία και προέκυψε αρχικά το 2004.

Το Ubuntu επικεντρώνεται στη χρηστικότητα, περιλαμβάνοντας εκτεταμένα τη χρήση της εντολής Root Sudo. Η εφαρμογή εγκατάστασης Ubiquity επιτρέπει την εγκατάσταση του Ubuntu στο σκληρό δίσκο από ένα περιβάλλον Live CD χωρίς να χρειάζεται η επανεκκίνηση του συστήματος για την εγκατάσταση, ενώ η εφαρμογή Wubi επιτρέπει την εγκατάσταση μέσα από περιβάλλον Microsoft Windows, κάνοντας έτσι πιο εύκολη τη μετάβαση στο Linux για τους χρήστες Windows. Το Ubuntu επιπλέον επικεντρώνεται στη προσιτότητα και διεθνοποίηση, για να προσεγγίσει περισσότερους χρήστες. Ακόμα, μια από τις βασικές διαφορές του με άλλες διανομές, όπως το Debian, είναι ότι επιτρέπει την εγκατάσταση πακέτων μη ελεύθερου λογισμικού για ορισμένους ειδικούς σκοπούς, όπως για παράδειγμα οδηγούς υλικού (*drivers*) για ορισμένα είδη υλικού (hardware) ή codecs για την αναπαραγωγή πολυμέσων. Από την έκδοση 5.04, η κωδικοποίηση UTF-8 είναι η στάνταρ κωδικοποίηση χαρακτήρων. Το προκαθορισμένο περιβάλλον εργασίας αποκαλείται *Human* και χαρακτηρίζεται από αποχρώσεις του καφέ και πορτοκαλί. Εκτός από τα προκαθορισμένα εργαλεία συστήματος και άλλες μικρές εφαρμογές, στο Ubuntu μπορεί κανείς να χρησιμοποιήσει αμέσως μετά την εγκατάσταση, μεταξύ άλλων, τις εφαρμογές: τη σουίτα γραφείου OpenOffice.org, τον περιηγητή ιστοσελίδων Firefox, το πρόγραμμα μηνυμάτων Empathy, το πρόγραμμα επεξεργασίας εικόνων GIMP, καθώς και την εφαρμογή επιφάνειας εργασίας Compiz fusion. Δεκάδες μικρά παιχνίδια χαρτιών και παζλ είναι προεγκατεστημένα, περιλαμβάνοντας Sudoku και σκάκι. Το Ubuntu έχει προρρυθμισμένα όλες τις θύρες επικοινωνίας κλειστές προσθέτοντας μεγαλύτερη ασφάλεια, μερικοί όμως χρήστες

προτιμούν τη χρήση κάποιου τείχους προστασίας (Firewall) για να παρακολουθούν τις εισερχόμενες και εξερχόμενες συνδέσεις.

Τώρα θα σας περιγράψω πως θα εγκαταστήσουμε τους μεταφραστές και πως λέγονται αυτοί για να μπορούμε να τους εκτελέσουμε στο τερματικό

Το ubuntu είναι ένα διαδραστικό λειτουργικό σύστημα στο οποίο μπορούμε είτε να εγκαταστήσουμε ένα λογισμικό μέσω ενός άλλου προγράμματος είτε άμεσα μέσω του τερματικού , το τερματικό είναι το αντίστοιχο command line/πίνακας εντολών των windows, σε μια ελληνικη εκδοση ubuntu αν πάμε στις εφαρμογές μετά στα βοηθήματα θα δούμε το τερματικό όπου θα πρέπει να το επιλέξουμε . Στο επίπεδο του τερματικού για να εγκαταστήσουμε ένα λογισμικό στα ubuntu πρέπει να δώσουμε την εντολή sudo apt-get install, είναι μια εντολή που πρέπει να δίνουμε πάντα όταν θέλουμε να εγκαταστήσουμε κάτι, ακολουθούμενη η λέξη που πρέπει να γράψουμε για να εγκαταστήσουμε το πρόγραμμα που επιθυμούμε στην περίπτωση μας είναι ο gcc μεταφραστής άρα η εντολή είνια η εξής :

sudo apt-get install gcc

Στις εκδόσεις ubuntu από 8.04 και πάνω για να εγκαταστήσεις ένα πρόγραμμα μπορείς να το κανείς μέσω του κέντρου λογισμικού που δεν χρειάζεται να ξέρεις εντολές που στις περισσότερες περιπτώσεις απωθεί τους χρήστες για να ασχοληθούν με τα λίνουξ.

Τώρα ας περιγράψουμε πιος είναι ο μεταφραστής gcc (gnu compiler collection)

Το **GNU Compiler Collection** είναι ένα compiler platform που υποστηρίζει διάφορες γλώσσες προγραμματισμού και έχει φτιαχτεί απ το GNU Project. Εκτός απο το να είναι ο επίσημος compiler του GNU Project , το GCC έχει γίνει επίσης ο αρχικός compiler στα περισσότερα Unix λειτουργικά συστήματα, όπως το GNU/Linux, την οικογένεια BSD, το Mac OS X και άλλα. Το GCC έχει γίνει ported σε μια μεγάλη γκάμα αρχιτεκτονικών και χρησιμοποιείται πολύ συχνά από εφαρμογές σε εμπορικά πακέτα και σε εφαρμογές κλειστού κώδικα. Ο compiler μπορεί να παράξει κώδικα για πάρα πολλές πλατφόρμες όπως το Playstation 2 και το Sega Dreamcast. Αρκετές εταιρίες απασχολούνται αποκλειστικά με το να παρέχουν ports του GCC για διάφορες πλατφόρμες και οι διάφοροι κατασκευαστές chip θεωρούν ένα port του GCC για την αρχιτεκτονική τους σχεδόν απαραίτητο για την επιτυχία της. Η εργαλειοθήκη του GCC υποστηρίζει πολλές γλώσσες. Αυτές περιλαμβάνουν τις: C, C++, Java,

Objective-C, G-Fortran, Objective-C++ και όχι μόνο. Άλλες γλώσσες που υποστηρίζονται είναι οι Pascal, Modula-3, GDC, Mercury και VHDL.

Αφού αρχικά αναλύσαμε μεχρι τωρα το λειτουργικο στο οποίο τρέχουμε τον κώδικα αλλά και τον μεταφραστή που πρέπει να εγκαταστήσουμε , θα αναλύσουμε τον κώδικα που στην περίπτωση μας είναι η γλώσσα προγραμματισμού c

Η C είναι μια διαδικαστική γλώσσα προγραμματισμού γενικής χρήσης η οποία αναπτύχτηκε στις αρχές της δεκαετίας 1970-1980 από τον Ντένις Ρίτσι στα εργαστήρια Bell Labs για να χρησιμοποιηθεί για την ανάπτυξη του λειτουργικού συστήματος UNIX. Απο τότε χρησιμοποιείται ευρύτατα, και ιδιαίτερα για ανάπτυξη προγραμμάτων συστήματος (system software) αλλα και για απλές εφαρμογές. Οι λόγοι της ραγδαίας ανάπτυξης της συγκεκριμένης γλώσσας προγραμματισμού είναι η ταχύτητα της, καθώς και το γεγονός ότι είναι διαθέσιμη στα περισσότερα σημερινά λειτουργικά συστήματα. Ανάμεσα στους σχεδιαστικούς στόχους που έπρεπε να καλύψει η γλώσσα περιλαμβανόταν το ότι θα μπορούσε να μεταγλωττιστεί άμεσα με τη χρήση single-pass compiler με άλλα λόγια, ότι θα απαιτούνταν μόνο ένας μικρός αριθμός από εντολές σε γλώσσα μηχανής για κάθε βασικό στοιχείο της, χωρίς εκτεταμένη run-time υποστήριξη. Ως αποτέλεσμα, είναι δυνατό να γραφτεί κώδικας σε C σε low level επίπεδο προγραμματισμού με ακρίβεια ανάλογη της συμβολικής γλώσσας, στην πραγματικότητα η C ορισμένες φορές αποκαλείται και χωρίς να υπάρχει πάντα αντιπαράθεση "high-level assembly" ή "portable assembly." Επίσης, γίνονται αναφορές στη C ως mid-level γλώσσα προγραμματισμού. Το 1983, το American National Standards Institute (ANSI) όρισε επιτροπή, τη X3J11, για να δώσει ένα σύγχρονο, πλήρη ορισμό της C. Μετά από μακρά και επίπονη επεξεργασία, το πρότυπο (standard) ολοκληρώθηκε το 1989 και επικυρώθηκε ως as ANSI X3.159-1989 "Programming Language C." Η συγκεκριμένη έκδοση της γλώσσας ονομάζεται συχνά ANSI C, ή ορισμένες φορές C89 (για να διαχωρίζεται από τη C99).

Το 1990, το πρότυπο ANSI για τη C (με ορισμένες μικρές τροποποιήσεις) υιοθετήθηκε από τον Οργανισμό Διεθνών Προτύπων (International Organization for Standardization (ISO)) ως ISO/IEC 9899:1990. Αυτή η έκδοση καλείται C90. Επομένως, οι όροι "C89" και "C90" αναφέρονται ουσιαστικά στην ίδια γλώσσα. από τους στόχους της διαδικασίας δημιουργίας του προτύπου ANSI για τη C ήταν να δημιουργήσει ένα υπερσύνολο της K&R C, το οποίο θα απορροφούσε πολλά χαρακτηριστικά που είχαν εισαχθεί στην πορεία. Παρόλα αυτά, η επιτροπή

συμπεριέλαβε και ορισμένα νέα χαρακτηριστικά, όπως function prototypes (δανεισμένα από τη C++), και ένα πιο ικανό προεπεξεργαστή. Η σύνταξη για τους ορισμούς παραμέτρων άλλαξε επίσης, ώστε να αντικατοπτρίζει το στυλ της C++.

Μετά τη διαδικασία καθορισμού του προτύπου ANSI, ο ορισμός της γλώσσας C παρέμενε σχετικά σταθερός για ορισμένο καιρό, ενώ η C++ συνέχιζε να αναπτύσσεται. Ωστόσο, το πρότυπο επανεξετάστηκε προς το τέλος της δεκαετίας του '90, γεγονός που οδήγησε στην έκδοση του ISO 9899:1999 το 1999. Το πρότυπο αυτό συχνά αναφέρεται ως "C99". Υιοθετήθηκε ως πρότυπο ANSI το Μάρτιο του 2000.

Ο GCC και μερικοί άλλοι C compilers υποστηρίζουν πλέον τα περισσότερα χαρακτηριστικά του C99. Ωστόσο, υπάρχει μικρότερη υποστήριξη από εταιρίες όπως η Microsoft και η Borland που εστίασαν περισσότερο στη C++, καθώς η C++ παρέχει παρόμοια λειτουργικότητα και συχνά ασύμβατους τρόπους (π.χ., η `complex` template class). Ο Brandon Bray από τη Microsoft είπε "Σε γενικές γραμμές, έχουμε δει μικρές απαιτήσεις για πολλά χαρακτηριστικά του C99. Μερικά χαρακτηριστικά έχουν μεγαλύτερη ζήτηση από άλλα, και θα τη λάβουμε υπόψιν μας σε μελλοντικές εκδόσεις εφόσον είναι συμβατά με τη C++."

Ακόμη και ο GCC με την εκτεταμένη υποστήριξη του C99 ακόμη δεν προσεγγίζει μια πλήρως συμβατή υλοποίηση, ορισμένα χαρακτηριστικά-κλειδιά λείπουν ή δεν λειτουργούν σωστά.

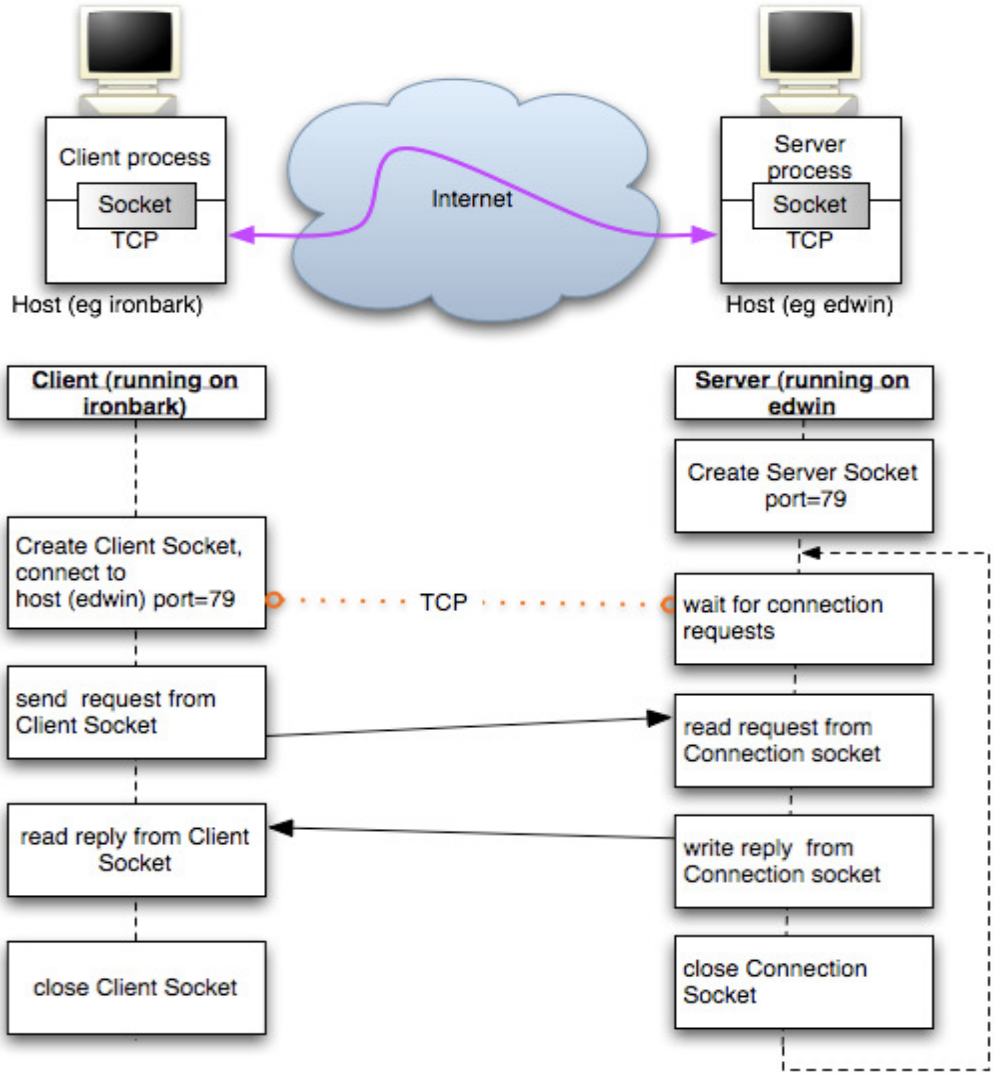
4.3 Ανάλυση για τον κώδικα του δέκτη και αποστολεα

Στην ακόλουθη ενότητα θα αναλύσουμε τους κώδικες του αποστολέα(sender) και του δέκτη(receiver). Πρωτού γίνει αυτό θα πρέπει να αναλύσουμε ένα θέμα που θα μας περιγράφει πως γίνεται η σύνδεση του αποστολέα με τον δέκτη. Για να επιτευχθεί η σύνδεση αυτή συμαντικό ρόλο παίζει το socket. Το socket είναι το κομμάτι του κώδικα που μας βοηθάει να πετύχουμε την σύνδεση του δέκτη με τον αποστολέα στο επίπεδο εφαρμογών.

Θα αναλύσουμε το socket στην γενική του ιδέα ,Μια εφαρμογή αποστολλέα ακούει κανονικά σε συγκεκριμένη θύρα αναμονής για τις αιτήσεις σύνδεσης από έναν δέκτη. Όταν μια αίτηση σύνδεσης φτάνει, ο δέκτης και ο αποστολλέας θα συγκροτήσουν μια ειδική σχέση επί της οποίας μπορούν να επικοινωνούν. Κατά τη διάρκεια της διαδικασίας σύνδεσης, ο δέκτης έχει έναν τοπικό αριθμό θύρας, και δεσμεύει την

υποδοχή σε αυτό. Ο δέκτης μιλά στο αποστολλέα γράφοντας στην θύρα και παίρνει πληροφορίες από το αποστολλέα από την ανάγνωσή του. Ομοίως, ο αποστολλέας παίρνει ένα νέο τοπικό αριθμό θύρας (χρειάζεται ένα νέο αριθμό θύρας ώστε να μπορεί να συνεχίσει να ακούει για αιτήσεις σύνδεσης στην αρχική θύρα). Ο αποστολλέας δεσμεύει επίσης μια υποδοχή στην τοπική θύρα της και επικοινωνεί με τον δέκτη με την ανάγνωση και εγγραφή σε αυτό. Ο δέκτης και ο αποστολλέας πρέπει να συμφωνήσουν σε ένα πρωτόκολλο - δηλαδή, πρέπει να συμφωνήσουν σχετικά με τη γλώσσα των πληροφοριών που μεταφέρονται πέρα δώθε μέσα στην υποδοχή.

Τελικά το socket είναι είναι ένα τελικό σημείο ενός συνδέσμου αμφίδρομης επικοινωνίας ανάμεσα σε δύο προγράμματα που εκτελούνται στο δίκτυο.



Σχήμα 5 Ανάλυση socket

Κεφάλαιο 4 σελίδα 38

Θα αναλύσουμε τον κώδικα σε επίπεδα έτσι ώστε η ανάλυση μας να είναι πιο κατανοητή και

περιεκτική. Όπως ξέρουμε η γλώσσα προγραμματισμού C είναι η γλώσσα η οποία στο επιπεδο εφαρμογών είναι στο χαμιλότερο επίπεδο και μιλάει με το φυσικό επιπεδο πιό άμεσα απο άλλες γλώσσες προγραμματισμού .Έτσι για να μπορούμε να έχουμε άμμεση επικοινωνία όλων των επιπέδων απο το φυσικό στο εφαρμογών χρησιμοποιούμε τις βιβλιοθήκες οι οποίες είναι κομμάτια κώδικα τα οποία μπορούμε να καλούμε άμμεσα απο το επιπεδο εφαρμογών χωρίς να χρειάζεται να γράψουμε τα αντίστοιχα προγράμματα για τα χαμιλότερα επίπεδα . Λοιπόν ας αναλύσουμε τι είναι το επίπεδο OSI , Το *μοντέλο OSI* είναι μια ιεραρχική δομή επτά επιπέδων που καθορίζει τις προδιαγραφές επικοινωνίας μεταξύ δύο υπολογιστών, ορίζοντας επακριβώς τον σκοπό κάθε επιπέδου αλλά και τα χρησιμοποιούμενα πρωτόκολλα, και τυποποιήθηκε ως πρότυπο ISO 7498-1. Θεωρήθηκε ότι θα επέτρεπε τη λειτουργική συνεργασία μεταξύ ποικίλων ψηφιακών συσκευών που ήταν διαθέσιμες στην αγορά. Το μοντέλο επιτρέπει σε όλα τα στοιχεία ενός δικτύου να συλλειτουργούν, με κάθε στοιχείο να υλοποιεί ένα ή περισσότερα πρωτόκολλα δικτύωσης, ανεξάρτητα από το ποιος είναι ο κατασκευαστής τους. Περί τα τέλη της δεκαετίας του 1980 ο ISO συνιστούσε την εφαρμογή του *μοντέλου OSI* ως κοινώς αποδεκτού υποδείγματος σχεδιασμού δικτύων.Ωστόσο εκείνη την εποχή η στοίβα πρωτοκόλλων TCP/IP, η οποία βασιζόταν σε ελαφρώς διαφορετική διαστρωμάτωση επιπέδων, ήταν ήδη επί πολύ καιρό σε ευρεία χρήση. Το TCP/IP ήταν θεμελιώδες για το δίκτυο ARPANET και τα άλλα δίκτυα που εξελίχθηκαν στο σημερινό Διαδίκτυο. Ως αποτέλεσμα το *μοντέλο OSI* παραμερίστηκε και σήμερα μόνο ένα υποσύνολό του χρησιμοποιείται ακόμη. Η επικρατούσα αντίληψη είναι ότι οι περισσότερες προδιαγραφές του είναι περίπλοκες και η πλήρης λειτουργικότητά του θα χρειαζόταν μεγάλο χρόνο κατασκευής, αν και συνεχίζουν να υπάρχουν σθεναροί υποστηρικτές του.

Περιγραφή των επιπέδων OSI

Επίπεδο 7: Εφαρμογών

Το *επίπεδο εφαρμογών* παρέχει στον χρήστη έναν τρόπο να προσπελάσει μέσω μιας εφαρμογής τις πληροφορίες ενός δικτύου. Αυτό το επίπεδο είναι η κύρια διασύνδεση του χρήστη με την εφαρμογή και, συνεπώς, με το δίκτυο. Στο επίπεδο αυτό γίνεται η διαχείριση των κατανεμημένων εφαρμογών, η αποστολή τουηλεκτρονικού ταχυδρομείου κλπ. Παραδείγματα πρωτοκόλλων επιπέδου εφαρμογών αποτελούν τα Telnet, FTP,SMTP και http.

Επίπεδο 6: Παρουσίασης

Το *επίπεδο παρουσίασης* μετασχηματίζει τα δεδομένα σε τυπική μορφή που την αναμένει το επίπεδο εφαρμογών. Στο επίπεδο αυτό τα δεδομένα υφίστανταικρυπτογράφηση,συμπίεση, κωδικοποίηση MIME και όποια άλλη διαμόρφωση απαιτεί η μορφή δεδομένων ή ο σχεδιαστής του πρωτοκόλλου. Παραδείγματα αποτελούν η μετατροπή αρχείων από κώδικα EBCDIC σε κώδικα ASCII και η μετατροπή της δομής των δεδομένων σε μορφή XML ή αντίστροφα.

Επίπεδο 5: Συνόδου

Το *επίπεδο συνόδου* ελέγχει τις συνόδους (δηλαδή τις ανταλλαγές δεδομένων) μεταξύ δύο υπολογιστών, του Α και του Β. Ξεκινά, διαχειρίζεται και τερματίζει τη σύνδεση μεταξύ μιας τοπικής και μιας απομακρυσμένης εφαρμογής. Αντιμετωπίζει λειτουργίες FDX *full duplex, οι Α και Β μιλούν ταυτόχρονα από δύο κανάλια* ή HDX *half-duplex, μιλάει ο Α και μετά απαντάει ο Β από το ένα διαθέσιμο κανάλι*, ενώ υποστηρίζει διαδικασίες αποθήκευσης κατάστασης, αναβολής, τερματισμού και επανεκκίνησης. Αυτό το επίπεδο είναι υπεύθυνο για το ομαλό κλείσιμο της συνόδου που είναι ιδιότητα του TCP και επίσης για την *αποθήκευση* και *ανάκτηση κατάστασης*, λειτουργίες οι οποίες δεν χρησιμοποιούνται στην στοίβα πρωτοκόλλων του Διαδικτύου.

Επίπεδο 4: Μεταφοράς

Το *επίπεδο μεταφοράς* διεκπεραιώνει τη μεταφορά των δεδομένων από χρήστη σε χρήστη, απαλλάσσοντας έτσι τα ανώτερα επίπεδα από κάθε φροντίδα να προσφέρουν αξιόπιστη μεταφορά δεδομένων από το ένα άκρο της επικοινωνίας στο άλλο. Το επίπεδο μεταφοράς ελέγχει την αξιοπιστία ενός χρησιμοποιούμενου καναλιού με έλεγχο ροής,κατάτμηση και αποτμηματοποίηση, καθώς και έλεγχο

σφαλμάτων.Ορισμένα πρωτόκολλα καταγράφουν καταστάσεις και συνδέσεις, οπότε κρατούν λογαριασμό των πακέτων και επανεκπέμπουν αυτά που δεν παρελήφθησαν

σωστά. Τα διάφορα πρωτόκολλα μορφοποιούν διαφορετικά τα εκπεμπόμενα πακέτα πληροφοριών, αλλά τα προς αποστολή δεδομένα παραλαμβάνονται αρχικά από τα ανώτερα επίπεδα.Το συνηθέστερο παράδειγμα πρωτοκόλλου μεταφοράς είναι το TCP. Άλλα πρωτόκολλα μεταφοράς είναι τα UDP, πρωτόκολλο για ασυνδεσμική αποστολή δεδομένων,SCTP, πρωτόκολλο ελέγχου της ροής μετάδοσης.

Επίπεδο 3: Δικτύου

Το *επίπεδο δικτύου* παρέχει τα λειτουργικά και διαδικαστικά μέσα για τη μεταφορά στοιχειοσειρών δεδομένων μεταβλητού μήκους από μια προέλευση σε έναν προορισμό, μέσα από ένα ή περισσότερα ενδιάμεσα δίκτυα, ενώ διατηρεί την ποιότητα εξυπηρέτησης που απαιτεί το επίπεδο μεταφοράς. Το επίπεδο δικτύου εκτελεί λειτουργίες δρομολόγησης, μεπιθανές κατατμήσεις αποτμηματοποιήσεις, και αναφέρει σφάλματα σχετικά με την παράδοση των πακέτων. Οι δρομολογητές λειτουργούν στο επίπεδο αυτό· διακινώντας δεδομένα σε διασυνδεδεμένα δίκτυα έκαναν το Διαδίκτυο πραγματικότητα. Υπάρχουν και δικτυακοί διακόπτες που σχετίζονται με τις διευθύνσεις IP. Εδώ υπάρχει μια λογική οργάνωση και τις τιμές των διευθύνσεων τις καθορίζει ιεραρχικά ο τεχνικός των επικοινωνιών. Το πλέον αναγνωρίσιμο παράδειγμα πρωτοκόλλου δικτύου είναι το Πρωτόκολλο Διαδικτύου.

Επίπεδο 2: Ζεύξης Δεδομένων

Το *επίπεδο ζεύξης* δεδομένων παρέχει τα λειτουργικά και διαδικαστικά μέσα για τη μεταφορά δεδομένων από μια συσκευή ενός τοπικού δικτύου σε άλλη, αλλά και για την ανίχνευση και διόρθωση σφαλμάτων που συμβαίνουν στο φυσικό επίπεδο. Οι μη ιεραρχημένες διευθύνσεις των συσκευών εδώ είναι οι φυσικές (π.χ. MAC διευθύνσεις), δηλαδή είναι προκαθορισμένες και αποθηκευμένες στις κάρτες δικτύου των επικοινωνούντων κόμβων από το εργοστάσιο. Το πιο γνωστό πρότυπο αυτού του επιπέδου είναι το Ethernet, για τοπικά δίκτυα. Άλλα παραδείγματα πρωτοκόλλων ζεύξης δεδομένων αποτελούν τα:

- HDLC και ADCCP, για συνδέσεις από-σημείο-σε-σημείο.
- 802.11, για ασύρματα τοπικά δίκτυα.

Στα τοπικά δίκτυα της οικογένειας πρωτοκόλλων IEEE 802, και σε κάποια άλλα όπως το FDDI, αυτό το επίπεδο μπορεί να διαιρεθεί σε δύο μικρότερα:

- Ένα επίπεδο ελέγχου πρόσβασης στο κοινό μέσο, το υποεπίπεδο MACl, Έλεγχος Πρόσβασης Μέσου

- Ένα ανώτερο επίπεδο ελέγχου λογικών συνδέσεων, το υποεπίπεδο LLCl, Έλεγχος Λογικών Ζεύξεων, όπου επικρατεί καθολικά το πρωτόκολλο IEEE 802.2 ανεξάρτητα από το υποκείμενο πρωτόκολλο MAC ή φυσικού επιπέδου.

Στο επίπεδο αυτό λειτουργούν οι δικτυακές *γέφυρες* και οι δικτυακοί διακόπτες. Η συνδεσιμότητα παρέχεται μόνο για κόμβους που συνδέονται στο ίδιο κοινό μέσο .

Επίπεδο 1: Φυσικό

Το *φυσικό επίπεδο* ορίζει όλες τις ηλεκτρικές και φυσικές προδιαγραφές της επικοινωνίας. Σ' αυτές περιλαμβάνονται οι σχηματισμοί των ακίδων, οι επιτρεπτές τάσεις, οι προδιαγραφές των καλωδίων κλπ. Συσκευές φυσικού επιπέδου είναι οι διανεμητές, οι επαναλήπτες, οι κάρτες δικτύου, οι προσαρμοστές διαύλου. Οι κυριότερες λειτουργίες και υπηρεσίες του φυσικού επιπέδου είναι:

- Έναρξη και τερματισμός της ηλεκτρικής σύνδεσης μιας επικοινωνιακής συσκευής.
- Συμμετοχή σε διαδικασίες όπου οι επικοινωνιακές συσκευές εξυπηρετούν αποτελεσματικά πολλούς χρήστες . Επιλύονται προβλήματα προτεραιότητας πρόσβασης και ελέγχου ροής δεδομένων.
- Διαμόρφωση και αποδιαμόρφωση των ψηφιακών δεδομένων κατά τη μετάδοση από συσκευή σε συσκευή. Για παράδειγμα, τα ψηφιακά ηλεκτρικά σήματα μπορεί να ταξιδέψουν ως αναλογικά σε χάλκινο καλώδιο, μετά σε οπτική ίνα, μετά να μεταδοθούν από ραδιοζεύξη ή δορυφορικά, να φθάσουν πάλι αναλογικά σε χάλκινο καλώδιο και να γίνουν ψηφιακά στον παραλήπτη.

Κώδικας Αποστολέα:Αφού αναλύσαμε τα γεγονότα που παίζουν ρόλο για την σωστή μετάδωση της πληροφορίας απο εναν υπολογιστή στον άλλο θα προχωρίσουμε στην ανάλυση του κώδικα και ιδιαίτερα στις βιβλιοθήκες όπως ανφέραμα αρχικά στην ενότητα . Στον κώδικα μας οι βασικές βιβλιοθήκες είναι οι εξείς:

```
#include <sys/types.h>   /* for type definitions */

#include <sys/socket.h>  /* for socket API function calls */

#include <netinet/in.h>  /* for address structs */
```



```
#include <arpa/inet.h>   /* for sockaddr_in */
```

```
#include <stdio.h>       /* for printf() */

#include <stdlib.h>      /* for atoi() */

#include <string.h>      /* for strlen() */

#include <unistd.h>      /* for close() */

#include <errno.h>

#include <net/if.h>

#include <netdb.h>

#include <stdint.h>

#include <stdbool.h>

#include <assert.h>
```

#include<stdio.h>:Η γλώσσα προγραμματισμού C παρέχει μια πρότυπη βιβλιοθήκη με συναρτήσεις για προσπέλαση αρχείων τύπου scanf Αυτές οι συναρτήσεις βρίσκονται στο αρχείο επικεφαλίδας `<stdio.h>`.

#include <stdlib.h>:Αυτή η κεφαλίδα καθορίζει ορισμένες γενικές λειτουργίες σκοπού, συμπεριλαμβανομένης της δυναμικής διαχείρισης μνήμης, γεννήτρια τυχαίων αριθμών, επικοινωνία με το περιβάλλον, ακέραιες μεταβλητές τιμές, αναζήτηση, ταξινόμηση και τη μετατροπή μεταβλητών τιμών.

#include <string.h>:Αυτό το αρχείο κεφαλίδας ορίζει διάφορες λειτουργίες για τον χειρισμό strings και arrays. Με τον χειρισμό των πινάκων και των strings χειριζόμαστε ουσιαστικά την μνήμη αφού διαχειραζόμαστε ουσιαστικά τα δεδομένα ,τέτοιες συναρτήσεις είναι η memset,strlen και άλλες συναρτήσεις.

#include <sys/types.h>: Όπως λέει και η επικεφαλίδα αυτή η βιβλιοθήκη μας επιτρέπει να καθορίσουμε το είδος του socket το οποίο θα αναλύσουμε, στην περιπτωση μας ένα socket μπορεί να μεταφέρει udp ή tcp πακέτα

#include <sys/socket.h>: Η βιβλιοθήκη αυτή μας επιτρέπει να καλέσουμε συναρτήσεις που έχουν να κάνουν με το socket. Τέτοιες συναρτήσεις είναι **getaddrinfo** που μας επιτρέπει να δώσουμε πληροφορίες για την διεύθυνση.Ακόμα είναι η **getnameinfo** που μετατρέπει την δυαδική αναπαράσταση μιας διεύθυνσης IP, με τη μορφή ενός struct sockaddr δείκτη σε συμβολοσειρές κειμένου που αποτελείται από το όνομα ή, αν η διεύθυνση δεν μπορεί να επιλυθεί σε ένα όνομα, σε αναπαράσταση διεύθυνσης IP, καθώς και το όνομα ή τον αριθμό θύρας. Η **freeaddrinfo** θα ελευθερώσει τη μνήμη που διατίθενται από την getaddrinfo συνάρτηση που έχει ως αποτέλεσμα να δημιουργήσει μια λίστα με συνδέσμους απο addrinfo structures όπου μέσω μιάς λούπας η συνάρτηση αυτή θα καταργεί κάθε σύνδεσμο.

#include <netinet/in.h>: Μπορούμε και καθορίζουμε το είδος της διεύθυνσης στην οποίοα ανοίκουμε μέσω του AF_INET και AF_INET6 για ipv4 και ipv6 αντίστοιχα, αλλά και σε ποιό πρωτόκολλο ανοίκουνε μέσω του PF_INET PF_INET6 ώστε να καθορίσουμε αν είναι TCP ή UDP πακέτα.

#include <arpa/inet.h>: Όπως είναι προφανές η βιβλιοθήκη αυτή μας βοηθά να μετατρέψουμε τις διαδυκές διευθύνσεις μέσω του πίνακα ARPA σε διευθύνσει με χαρακτήρες γράμματα.

#include <unistd.h>:ορίζει διάφορες συμβολικές σταθερές και είδη, και δηλώνει διάφορες λειτουργίες.

#include <errno.h>: Ορίζει μακροεντολές για να αναφέρει τις συνθήκες σφάλματος, όπου αποθηκεύονται σε μια τοποθεσία που ονομάζεται στατική errno.Η τιμή αποθηκεύεται στο errno από ορισμένες λειτουργίες της βιβλιοθήκης, όταν εντοπίσουν τα λάθη. Κατά την εκκίνηση του προγράμματος, η αποθηκευμένη τιμή είναι μηδενική. Όπου λειτουργεί μόνο τιμές μεγαλύτερες από το μηδέν. Κάθε λειτουργία της βιβλιοθήκης μπορεί να αλλάξει την αποθηκευμένη τιμή πριν από την επιστροφή, έστω και αν δεν ενοπίσουν σφάλματα. Οι περισσότερες λειτουργίες αναφέρουν ότι

εντόπισε ένα σφάλμα κατά την επιστροφή ιδιαίτερης τιμής , συνήθως NULL για τις λειτουργίες απόδοσης, και -1 για τις λειτουργίες που επιστρέφει ακέραια τιμή. Μερικές λειτουργίες απαιτούν τον καλούντα να προκαθορίσει το errno στο μηδέν .Ο κωδικός λάθους μακροεντολής επεκτείνεται σε μια τιμή τύπου ακέραιος, που περιέχει τον τελευταίο κωδικό σφάλματος που δημιουργείται σε κάθε συνάρτηση με την errno εγκατάσταση.

#include <net/if.h>: Η βιβλιοθήκη αυτή μας επιτρέπει να καλέσουμε τιμές που δεν έχουν πρόσυμο τις γνωστής απρόσυμες τιμές/unsigned οι οποίες έχουν μεγαλύτερο εύρος τιμών, απο ότι οι ακέραιες(integer) ή πραγματικές(float) τιμές.

#include <netdb.h>: Η βιβλιοθήκη αυτή μας επιτρέπει να κραταμε μέσω μιάς sruckt μεταβλητές τιμές για τον τύπο της θύρα και τον τύπο της διεύθυνσης.

#include <stdint.h>: Η βιβλιοθήκη αυτή δηλώνει σύνολα ακεραίων τύπων που έχουν συγκεκριμένο πλάτος και καθορίζει αντίστοιχα σύνολα μακροεντολών. Καθορίζει επίσης τις μακροεντολές που καθορίζουν τα όρια των ακεραίων τύπων που αντιστοιχούν σε τύπους που ορίζονται από άλλα πρότυπα.

#include <stdbool.h>: Το πρότυπο C99 περιλαμβάνει ένα νέο τύπο δεδομένων που ονομάζεται «Bool», για σύντομία του Boolean .Διαθέσιμη στο πρότυπο C99 είναι το αρχείο« stdbool.h , η οποία καθορίζει δύο σχετικές λέξεις-κλειδιά: bool, τύπο δεδομένων, αλήθεια που είναι ένα ψευδώνυμο για 1 και ψευδή που είναι ένα ψευδώνυμο για 0.

#include <assert.h>: είναι ένα αρχείο κεφαλίδας στην πρότυπη βιβλιοθήκη της γλώσσας προγραμματισμού C που ορίζει η μακροεντολή. Η μακροεντολή εφαρμόζει έναν ισχυρισμό, ο οποίος μπορεί να χρησιμοποιηθεί για την επαλήθευση παραδοχών που έγιναν από το πρόγραμμα και να εκτυπώσετε ένα διαγνωστικό μήνυμα, αν η υπόθεση αυτή είναι εσφαλμένη. Όταν εκτελεστεί, αν η έκφραση είναι ψευδής δηλαδή, συγκρίνει ίση με 0, γράφει πληροφορίες σχετικά με την πρόσκληση που απέτυχε στην stderr και στη συνέχεια καλεί ματαίωση . Οι πληροφορίες που γράφει για το stderr περιλαμβάνει: το όνομα του αρχείου πηγής (η προκαθορισμένη FILE__ __ μακρο), ο αριθμός γραμμής πηγή (το προκαθορισμένο __ μακρο LINE__), η λειτουργία πηγής (το προκαθορισμένο αναγνωριστικό __ func__)

Όπως αναλύουμε και περιγράφουμε στο κεφάλαιο αυτό τον κώδικα του δέκτη και του αποστολέα ο σκοπός μου όπως αναφέρω και στην αρχή δεν είναι απλα να αναφέρω τα βασικά στοιχεία του κώδικα αλλα μια λεπτομερής ανάλυση έτσι ώστε ακόμα και ένας που δεν έχει ασχοληθεί με τον προγραμμάτισμό να καταλάβει τις βασικές αρχές της γλώσσας προγραμματισμού C

Αναλύοντας παραπάνω τον ρόλο των βιβλιοθηκών και ποιές βιβλιοθήκες είναι αυτές που χρειαζόμαστε για την επίλυση του κώδικα , θα μεταβούμε στο κύριο μέρος του κώδικα .Ένα απλό πρόγραμμα στη γλώσσα C αποτελείται απο τρία βασικά τμήματα μια κύρια συνάρτηση με την ονομασία main()

```
main()

{

δηλώσεις μεταβλητών;

εκτελέσιμες προτάσεις;

}
```

Δηλώσεις μεταβλητών: Όλες οι μεταβλητές που χρησιμοποιούνται σε ένα πρόγραμμα πρέπει να έχουν δηλωθεί προηγουμένως. Μια δήλωση μεταβλητής περιλαμβάνει την αρχική της τιμή ,τον τύπο της και το όνομα της . Ο τύπος των μεταβλητών μπορεί να είναι ακέραιος int ,δυαδικός float και χαρακτήρας char.

```
Int a=4;

float b=3.0;

char charaktires ;
```

Εκτελέσιμες προτάσεις: Καθώς η γλώσσα C είναι μια γλώσσα που διαβάζει γραμμή γραμμή τον κώδικα απο πάνω μέχρι κάτω το πρώτο πράγμα που διαβάζει είναι η συνάρτηση main(). Μέσα στην υπάρχουσα συνάρτηση μπορούμε να γράψουμε και άλλες συναρτήσεις στις οποίες θα εκτελούνται εντολές .Οι εντολές αυτές μπορεί να είναι αριθμυτικές πράξεις (-,+,*,/) αλλά και λογικές πράξεις , όταν ονομάζουμε λογικές πράξεις εννοούμε τις πράξεις αυτές που επαληθεύουν αν μια πράξη είναι

αληθης 1 ή ψευδής 0 οι τελεστές που μας βοηθάνε στην πράξη αυτή είνοι οι (<,>,<=,=>,==,!=,&&,||) μικρότερο,μεγαλύτερο,μικρότερο και ισον,μεγαλύτερο και ίσον ,ισσοτητα,διάφορο ,λογικό AND(ΚΑΙ) και λογικό OR(Η).Στις λογικές πράξεις πάντα υπερισχύει η ψευδής οταν έχουμε την λογική πραξη ΚΑΙ δηλαδή αν έχουμε ΑΛΗΘΗΣ ΚΑΙ ΨΕΥΔΗΣ=ΨΕΥΔΗΣ. Ενώ αν έχουμε λογική πράξη Η υπερισχύει η ΑΛΗΘΗΣ, δηλαδή ΑΛΗΘΗΣ Η ΨΕΥΔΗΣ=ΑΛΗΘΗΣ.Βασικές εντολές με τις οποίες εμφανίζουμε ενα μήνυμα ή δίνουμε εμείς μια τίμή στο πρόγραμμα διαδραστικά μέσω του πληκτρολογίου και όχι αρκικοποιόντας την απο τον κώδικα είναι η printf και η scanf αντίστοιχα .

printf("dwse timi gia to a:",a); ------> scanf("%d",&a);

να αναφέρουμε ότι μεταβλητές μέσα σε αυτες τις εντολές δηλόνοναι με έναν συγκεκριμένο χαρακτήρα για τις ακέραιες είναι το d,για τις δεκαδικές f και για τους χαρακήρες c. Ακόμα υπάρχουν οι αλφαριθμυτικές τιμές που ονομάζονται string και παίρνουν τιμές αριθμούς αλλά και χαρακτήρες και χαρακτηρίζοντια με τον χαρακτήρα s. Τον κώδικα θα τον χωρίσουμε σε κομμάτια έτσι ώστε να είναι πιο κατανοητός έτσι ώστε να αναλύσουμε και πιο περιεκτικά και του υπόλοιπους κανόνες που πρέπει να ξέρουμε για την επίλυση του κωδικα.

```
int main (int argc,char **argv)

{

int error;

int s;

int q,w,e,r;

unsigned char buf1 [49152];

unsigned char buf2 [49152];

unsigned char buf3 [49152];

unsigned char buf4 [49152];
```

```
size_t len;

FILE * fp;

unsigned long cnt,im_row,im_col;

unsigned char tc;

unsigned int w1,w2,width,height,h1,h2,offset,bpp,j;

unsigned int padd,i,l;

unsigned char image[256][256][3];

unsigned char tmpim[196608];

struct addrinfo hint, *res;

struct addrinfo *multi;
```

Αρχικά δώσαμε τις βιβλιοθήκες που πρέπει να δώσετε για να εκτελεστέι το πρόγραμμα τώρα σας δείνω την συνάρτηση mai() και τις αρχικές μεταβλητές που πρέπει να δηλώσουμε . Για την μετάδωση της εικόνας απο τον αποστολλέα στον δείκτη ουσιαστικά πρέπει να μεταφέρουμε κάπιες θέσεις μνήμης που περιέχουν τα εικονοστηχεία απο τον ένα υπολογιστή στον άλλο , στην C η δέσμευση μνήμης γίνεται μέσω των πινάκων και ο έλεγχος των εικονστηχείων δηλάδή των μεταβλητών μέσα στον πίνακα γίνεται με τους δείκτες *. Η συνάρτηση main() μπορεί να έχει δύο το πολύ συγκεκριμένες παραμέτρους οι οποίες παίρνουν τιμές απο την γραμμή εκτέλλεσης του προγράμματος στο περιβάλλον του λειτουργικού συστήματος. Οι παράμετροι αυτοί μπορεί να είναι οι argc,argv[]. Η argc μεταβιβάζει στην main() τον αριθμό των παραμέτρων με τις οποίες καλείται το πρόγραμμα . Η τιμή της argc είναι τουλάχιστον 1 δεδομένου ότι στις παραμέτρους υπολογίζεται και το όνομα του προγράμματος. Η argv[] είναι ένας δείκτης σε ένα πίνακα που περιέχει δείκτες σε char , η πρωτη θέση μνήμης του πίνακα argv[] δείχνει σε ένα σύνολο χαρακτήρων το οποίο περιέχει την απόλυτη διαδρομή του εκτελέσιμου αρχείου του προγράμματος . Οι υπόλοιπες θέσεις μνήμης δείχνουν σε σύνολα χαρακτήρων που περιέχουν τις παραμέτρους με τις οποίες έχει κληθεί απο την γραμμή εντολών το πρόγραμμα . Για αυτό η συνάρτηση main() περιέχει τις παραμέτρους αυτές :

```
main(argc,*argv[])
```

Οπως είπαμε η δήλωση μεταβλητών σε ένα πρόγραμμα χρειάζονται για αρχικοποίηση ή δήλωση τιμών για χρήση τους αργότερα σε μια συνθήκη. Τετοιες είναι και οι ακόλουθες:

int error; , int s; , int q,w,e,r; , unsigned int w1,w2,width,height,h1,h2,offset,bpp,j;, unsigned long cnt,im_row,im_col; unsigned char tc; unsigned int padd,i,l;

Η unsigned char είναι μια μεταβλητή όπως και οι άλλες που υπάρχουν στη συγεκριμένη γλώσσα προγραμματισμού με την διαφορά ότι παίρνει τιμές χωρίς πρόσιμο δηλαδήη οχι ανηρτικές. Η μεταβλητή int έχει μέγεθος 16 bit όπως και η float η char 1 bit .Υπάρχουν και οι μεταβλητές που καλούνται double ή long και έχουν μεγαλύτερο εύρως τιμών απο ότι οι κλασικές μεταβλητές και φυσικά μεγαλύτερο μέγθος μνήμης που είναι 32 bit. Συγκεκριμένα η char παίρνει 256 τιμές απο το -128 -128 ενώ η unsigned char απο το 1-256.

Στην πληροφορική υπάρχει η ορολογία buffer που είναι το κομμάτι της μνήμης που παίζει συμαντικό ρόλο για την διακίνηση της πληροφορίας της μνήμης . Οσο μεγαλύτερο buffer έχουμε τόσο καλύτερες επιδόσεις στη μνήμη έχουμε . Όπως προήπαμε στην C χρήση των μνημών γίνεται μέσω των πινάκων και επειδή κάθε μεταβλητή έχει μια θέση μνήμης με τον όρισμό των πινάκων μπορούμε να ορίζουμε το ακριβές μέγεθος την μνήμης που έχει μια μεταβλητή. Στην περίπτωση μας το μέγεθος είναι συγκεκριμένο γιατί μεταδίδουμε udp πακέτα και τα πακέτα αυτά που ουσιαστικά είναι ένας πίνακας buffer έχει μέγεθος το πολύ 65.000 bits. Όμως η εικόνα που μεταδήδουμε είναι 256*256*3 λογώ των τριών χρωμάτων κοκκινο πράσινο μπλέ και των 256 εικονοστηχείων οριζόντια και κάθετα έχει μέγεθος 196608 πρέπει να την χωρίσουμε σε 4 πίνακες για να την στήλουμε που θα έχουν μέγεθος 49152.

unsigned char buf1 [49152];

unsigned char buf2 [49152];

unsigned char buf3 [49152];

unsigned char buf4 [49152];

Ακόμα δημιουργούμε έναν πίνακα για όλη την εικονα με το μεγεθός της unsigned char tmpim[196608]; .Όπως και έναν πίνακα image για την διαχείριση της εικόνας με τα χρώματα και την σωστή τους διάταξη unsigned char image[256][256][3]; .

Στη C εκτώς απο την κλασική δήλωση μεταβλητών έχουμε την δυνατότητα να δηλώσουμε μεταβλητές όλων των τύπων αναφερόμενες με ένα κοινό όνομα. Μια δομή (struct) είναι συνήθως μια λογική ενότητα απο πληροφορίες που σχετίζονται μεταξύ τους . Για παράδειγμα το όνομα η διεύθυνση το τηλέφωνο και η ηλικία.

struct addrinfo hint, *res; ,struct addrinfo *multi;

Στην συγκεκριμένη περίπτωση έχουμε δημιουργήσει δομές που ασχολούνται με την διεύθυνση και την ταυτοποιήση της αλλά και για το multicasting .

```
memset( &hint, 0, sizeof( hint ) );

hint.ai_family = AF_INET6;

hint.ai_socktype = SOCK_DGRAM;

hint.ai_protocol = 0;

int count=0;

error = getaddrinfo( "ff02::1", NULL, &hint, &res );

if( error != 0 ) {

   perror( "getaddrinfo" );

}

struct sockaddr_in6 * addr = (struct sockaddr_in6*)res->ai_addr;

addr->sin6_port = htons( 7890 );

addr->sin6_scope_id = 2; // 2 happens to be the interface ID

s = socket( AF_INET6, SOCK_DGRAM, 0 );
```

Ο κώδικας που ακολούθησε είναι ο βασικός του socket . Η εντολή memset ρυθμίζει τα πρώτα bytes ενός μπλόκ μνήμης . Μετα ακολουθεί η πιστοποίηση της οικογένειας δικτυακού πρωτοκόλλου αλλά και στο udp πακέτο , ακόμα την

διεύθυνση του ipv6, το port και ID .Ακόμα εκτελει βασικές συνθήκες για την πιστοποίηση του πρωτοκόλλου και το είδος στο πακέτου στο οποίο ανοίκει .

Για να κάνουμε τον κώδικα πιο διαδραστικό με τον χρήστη δίνουμε εντολές printf έτσι ώστε να αποφασίζουμε πιες εικόνες θα στήλουμε .

```
printf("STEILE TIS EIKONES POU THELEIS\n");
printf("pata A gia na steileis tin prwti eikona ANEMOS\n",ch);
printf("pata B gia na steileis tin deuteri eikona ANEMOS1\n",ch);
printf("pata C gia na steileis tin triti eikona ANEMOS2\n",ch);
printf("pata D gia na steileis tin tetarti eikona ANEMOS3\n",ch);
printf("pata E gia na steileis tin pempti eikona ANEMOS4\n",ch);
printf("pata F gia na steileis tin ekti eikona ANEMOS5\n",ch);
printf("pata G gia na steileis tin evdomi eikona ANEMOS6\n",ch);
printf("pata H gia na steileis tin ogdoi eikona ANEMOS7\n",ch);
printf("pata I gia na steileis tin enati eikona ANEMOS8\n",ch);
printf("pata J gia na steileis tin dekati eikona ANEMOS9\n",ch);
```

Αμέσως μετά ανοίγουμε τις εικόνες με την σειρά που θέλουμε μέσω μιας συθήκης που ονομάζεται switch και μπορούμε να έχουμε την δινατότητα επιξεργασίας απο το πληκτρολόγιο .Μέσω ενός μετριτή μετράμε τις είκονε που στέλνουμε κάθε φορά .Κάθε φορά που ανοίγουμε μια εικόνα μας εμφανίζεται μύνυμα πιά εικόνα ανοίγουμε για να στήλουμε. Αν προσπαθήσουμε να ανοίξουμε μια εικόνα που δεν είναι bmp format εμφανιζεται ένα μήνυμα οτι δεν είνια ικανό το πρόγραμμα να την ανοίξει . Όλη αυτή η συνθήκη υπάρχει μέσα σε μια άλλη συνθήκη η οποία επαναλαμβάνεται για όλο το σύνολο των εικόνων .Έχουμε αριθμύσει τις εικόνες με κάπιες μεταβλητες για αυτό η συνθήκη τρέχει όσο πατήσουμε μια τιμή που δεν αληθεύει την συνθήκη .

```
do
{
switch(ch)
```

```
{
case 'A':
{
if (!(fp=fopen("Anemos.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos.bmp");
}
count++;
printf("stelnetai o Anemos\n");
break;
}
case 'B':
{
if (!(fp=fopen("Anemos1.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos1.bmp");
}
count++;
printf("stelnetai o Anemos1\n");
break;
}
case 'C':
{
if (!(fp=fopen("Anemos2.bmp","rb")))
```

```
{
        printf("Unable to load bin file: %s...\n","Anemos2.bmp");
}
count++;
printf("stelnetai o Anemos2\n");
break;
}
case 'D':
{
if (!(fp=fopen("Anemos3.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos3.bmp");
}
count++;
printf("stelnetai o Anemos3\n");
break;
}
case 'E':
{
if (!(fp=fopen("Anemos4.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos4.bmp");
}
count++;
printf("stelnetai o Anemos4\n");
```

```
break;
}
case 'F':
{
if (!(fp=fopen("Anemos5.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos5.bmp");
}
count++;
printf("stelnetai o Anemos5\n");
break;
}
case 'G':
{
if (!(fp=fopen("Anemos6.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos6.bmp");
}
count++;
printf("stelnetai o Anemos6\n");
break;
}
case 'H':
```

```
{
if (!(fp=fopen("Anemos7.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos7.bmp");
}
count++;
printf("stelnetai o Anemos7\n");
break;
}
case 'I':
{
if (!(fp=fopen("Anemos8.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos8.bmp");
}
count++;
printf("stelnetai o Anemos8\n");
break;
}
case 'J':
{
if (!(fp=fopen("Anemos9.bmp","rb")))
{
        printf("Unable to load bin file: %s...\n","Anemos9.bmp");
```

```
}
count++;
printf("stelnetai o Anemos9\n");
break;
}}
```

Μετά ακολουθεί η συμπίεση της εικόνας που ανοίγουμε ,μέσω της εντολής feof τρέχουμε την αρχείο που ανοίγουμε απο το πρώτο ψηφίο μεχρι το τελευταίο απο πανω μεχρι κάτω. Όπου οριοθετούμε το οριζόντιο και κάθετο μήκος της εικόνας. Μετά τρέχουμε έναν διπλό πίνακα που έχει τις διαστάσεις της εικόνας στον οποίο αρχικοποιούμε στον πίνακα tmpim την συμπίεση της εικόνας για τα τρία χρώματα μπλέ ,κόκκινο και πράσινο . Αφού κάνουμε αυτό τρέχουμε έναν διπλό πίνακα στις διαστάσεις της εικόνας όπου αρχκοποιούμε τον πίνακα με τις διαστάσεις και τα χρώματα με τον πίνακα με τα συμπιεσμένα στοιχεία. Κατά την συμπίεση της εικόνας και την αρχικοποιήση της με τον πίνακα tmpim μέσω της 0x0ff μπορούνε να διαχωρίσουμε τα τρία βασικά χρώματα .

```
width  = w1+w2*256;
height = h1+h2*256;
int  BMPRES_X = height;
int  BMPRES_Y = width;
padd = 0;
printf("W:%d, H:%d Image Data Starts at:%d with %d bits/pixel \n",width,height,offset,bpp);
for (im_row=0;im_row<height;im_row++)
{
 for (im_col=0;im_col<width;im_col++)
 {
    tc = fgetc(fp) & 0x0ff;
    tmpim[(im_col*height+height-im_row-1)*3+2] = tc;
```

```
      if ((im_col%32==0)&&(im_row%32==0)) printf("%02u",tc/10);
            tc = fgetc(fp) & 0x0ff;
            tmpim[(im_col*height+height-im_row-1)*3+1] = tc;
            if ((im_col%32==0)&&(im_row%32==0)) printf("%02u",tc/10);
            tc = fgetc(fp) & 0x0ff;
            tmpim[(im_col*height+height-im_row-1)*3+0] = tc;
            if ((im_col%32==0)&&(im_row%32==0)) printf("%02u-",tc/10);
         }
         if (im_row%32==0) printf("\n");
         for (i=0;i<padd;i++)
         {
         tc = fgetc(fp) & 0x0ff;
            }
   }
fclose(fp);
for (i=0; i<height; i++) {
         for (j=0; j<width; j++) {
            image[i][j][2] = (int)(tmpim[(j*height+height-i-1)*3+2]); // blue
            image[i][j][1] = (int)(tmpim[(j*height+height-i-1)*3+1]); // green
            image[i][j][0] = (int)(tmpim[(j*height+height-i-1)*3+0]);// red
         } // end of line
}
```

Όπως πρόειπα το μέγεθος της εικόνας δεν μας επιτρέπει να την στήλουμε ολόκληρη για αυτό το λόγω θα κόψουμε την εικόνα σε κομμάτια και θα τα αρχικοποιήσουμε σε buffer για να τα στήλουμε σταδικά . Αυτό θα το κάνουμε δημιουργόντας έναν δισδιάστατο πίνακα όπου θα τον τρέχουμε για τις διαστάσεις της εικόνας στο μήκος

θα τρέχουμε για καθε μνήμη το 256/4 που ισσούται με 64 άρα το μήκος θα ισσούται με 64 και το ύψος με 256 και έτσι χωρίζουμε την εικόνα σε 4 κομμάτια .

```
for(i=1; i<=64; i++)
{
        for(j=1; j<=256; j++)
{
                q=0;
                buf1[q] = image[i][j][2];
                        q++;
                buf1[q] = image[i][j][1];
                        q++;
                buf1[q] = image[i][j][0];
                        q++;
}
}
printf("1st BUFFER OK...\n");
scanf("%d",&tmp);
w=0;
for(i=65; i<=128; i++)
{
        for(j=1; j<=256; j++)
        {
                        buf2[w] = image[i][j][2];
                        w++;
                        buf2[w] = image[i][j][1];
```

```
                    w++;
                    buf2[w] = image[i][j][0];
                   w++;
        }
}
printf("2nd BUFFER OK...\n");
scanf("%d",&tmp);
e=0;
for(i=129; i<=192; i++)
{
        for(j=1; j<=256; j++)
        {
                    buf3[e] = image[i][j][2];
                    e++;
                    buf3[e] = image[i][j][1];
                    e++;
                    buf3[e] = image[i][j][0];
                    e++;
        }
}
printf("3rd BUFFER OK...\n");
scanf("%d",&tmp);
r=0;
for(i=193; i<=256; i++)
{
```

```
        for(j=1; j<=256; j++)
        {
                        buf4[r] = image[i][j][2];

                        r++;
                        buf4[r] = image[i][j][1];
                        r++;
                        buf4[r] = image[i][j][0];
                        r++;
        }
}
```

Δύο απο τις βασικές συνθήκες που ισχύουν σε ένα socket είναι η sendto και η receivto . Συγκεκριμένα στη sendto ελέγχοντας κάπιες μεταβλητές όπως το μέγεθος του buffer το όνομα του buffer μια μεταβλητή που συνδέεται με την συνθήκη socket το μέγεθος της διεύθυνσης ip αλλά και η ίδια η διεύθυνση παίζουν συμαντικό ρόλο για να στήλουν το αρχείο που θέλουμε . Λόγω των τεσσάρων buffer έχουμε και τέσσερις συνθήκες sendto. Όσο δεν υπάρχει σύνδεση μεταξύ του αποστολέα με τον δέκτη για να εξακρίβώσει την διεύθυνση του δηλαδή το πακέτο acknowlegmet και να προχωρίσει στην μεταφορά του πακέτου πληροφορίας μας εμφανίζει ένα μήνυμα οτι δεν στάλθηκε ο buffer και αφού δεν στάλθηκε ο πρώτος buffer τερματίζεται το πρόγραμμα, το μήνυμα της μή αποστολής του κάθε buffer επαλυθεύεται απο την μια συνθήκη η οποία ελλέγχει αν ο buffer είναι μικρότερος και ίσος του 49152 αν επαλυθεύεται τότε εμφανίζει ένα μήνυμα που λέει οτι ο buffer στάλθηκε . Υπό τις συνθήκες που σταλούν και οι τέσσερις buffer εμφανίζεται ένα μήνυμα που μας λέει να στήλουμε τις εικόνες που θέλουμε ή να πατίσουμε μια μετβλητή στην περιπτωση μας X για να τερματίσουμε την αποστολή . Στην περίπτωση που πατήσουμε μια τιμή που είναι διαφορετική των τιμών που θέλουμε για να στηλουμε τις εικόνες πάλι τερματίζεται το πρόγραμμα . Απο τον μετριτή που είπαμε και παραπάνω μας εμφανίζει ένα μήνυμα με το σύνολο των εικόνων που στήλαμε.

```
if( sendto( s, buf1, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
```

```
        printf( "Error sending buf1\n" );

        printf(" afou dn stalthike o protos buffer tote diakoptete i apostoli\n");

        exit(1);

    } else printf("Buf1 OK! \n",buf1);
if( sendto( s, buf2, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
        printf( "Error sending buf2\n" );
    } else printf("Buf2 OK!\n");
if( sendto( s, buf3, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
        printf( "Error sending buf3\n" );
    } else printf("Buf3 OK!\n");
if( sendto( s, buf4, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
        printf( "Error sending buf4\n" );
    }else printf("Buf4 OK!\n");
printf("Mexri twra exoume stilei %d eikones\n",count);
printf("pata ena gramma  apo A-J gia tin apostoli twn eikonwn i pata X gia to
kleisimo:\n");
scanf("%s",&ch);
}while(ch<='J');
exit(1);
}
```

Αυτή ήταν και η ανάλυση του κώδικα του αποστολέα , αμέσως μετά θα αναλύσουμε τον κώδικα του δεκτη και στο τέλος θα σας παραθέσουμε και τους δύο κώδικες ολοκληρομένους .

Κώδικας Δέκτη: Όπως και ο κώδικας του αποστολλέα έτσι και του δέκτη σε κάπια μέρη του δεν εχουν διαφορές στον τρόπο που δηλώνουμε τις μεταβλητές αφού είναι οι ίδιες ή παρόμοιες έτσι σε κάθε περίπτωση ό,τι μέρη του κώδικα δεν αναλύσουμε θα συμαίνει ό,τι τα έχουμε ήδη περιγράψει και πριν .

Ουσιαστικά σε μια επικοινωνία δύο υπολογιστών ώστε να μεταφερθεί ένα αρχείο απο τον έναν στον άλλον βασικό ρόλο παίζουν τα επίπεδα του OSI τα οποία μας περιγράφουν πώς μια εικόνα απο το επίπεδο εφαρμογών περνά απο ολα τα επίπεδα φτάνει στο κάλώδιο ethernet και απο εκέι απο τον δρομολογητή στον αποδέκτη και τελικά αφου περάσει απο τα επίπεδα του OSI ξανά φτάνει στο τελευταίο επίπεδο αυτό των εφαρμογών και τελικά εμφανιζεται στην οθόνη μας . Με βασικό παράγωντα να παίζει ο τρόπος με το οποίο ελέγχεται η διεύθυνση δικτύου και η θύρα (port) όλα αυτά γίνονται και εκφράζοντια στο επίπεδο του socket όπου θα περιγράψουμε διεξώδικα δίνοντας σας και τον κώδικα .

```
memset( &hint, 0, sizeof( hint ) );

hint.ai_family = AF_INET6;

hint.ai_socktype = SOCK_DGRAM;

error = getaddrinfo( "ff02::1", NULL, &hint, &res );

if( error != 0 ) {

   perror( "getaddrinfo" );

   exit( 1 );

}

struct sockaddr_in6 * addr = (struct sockaddr_in6*)res->ai_addr;

addr->sin6_addr = in6addr_any;
```

```
addr->sin6_port = htons( 7890 );

s = socket( AF_INET6, SOCK_DGRAM, 0 );

if(bind( s, (struct sockaddr*) addr, res->ai_addrlen ) != 0 ) {

close( s );

perror( "bind" );

exit( 1 );

}

if( getaddrinfo( "ff02::1", NULL, &hint, &multi ) != 0 ) {

   close( s );

   perror( "getaddrinfo" );

}

struct ipv6_mreq mreq;

memset( &mreq, 0, sizeof(mreq) );

memcpy( &mreq.ipv6mr_multiaddr, &((struct sockaddr_in6 *) multi->ai_addr)->sin6_addr, sizeof(mreq.ipv6mr_multiaddr) );

mreq.ipv6mr_interface = 2; // 2 happens to be the interface ID; I've tried other values here

freeaddrinfo( multi );

if( setsockopt( s, IPPROTO_IPV6, IPV6_JOIN_GROUP, &mreq, sizeof(mreq) ) !=0 )

 {

   close( s );
```

```
  perror( "IPV6_JOIN_GROUP" );

}
```

Όπως γνωρίζουμε η γλώσσα προγραμματισμού C είναι μια ειδηκή γλώσσα με την οποία μπορούμε να διαχειριζόμαστε την μνήμη αφού λειτουργεί στο χαμηλότερο επίπεδο εφαρμογών απο όλες τις άλλες γλώσσες έτσι μέσω την συνθήκης memset που υπάρχει στην γλώσσα αυτή μπορούμε να ανανεώνουμε το μέγεθος των μπλοκών μνήμης .Έτσι κάθε φορά που δεχόμαστε πακέτα τα αναεώνουμε και μετά ελέγχουμε το σε ποιά κατηγορία ανοίκουν udp ή tcp σε πιό πρωτόκολλο δικτύου ipv4 ή ipv6 ποιά διεύθυνση χρησιμοποιούμε , αν ο αποστολέας δεν έχει στήλει πακέτο με την διεύθυνση στην οποία ανοίκει στο δύκτιο μας εμφανίζει μήνυμα να δώσουμε διεύθυνση έτσι ώστε να κατευθύνουμε το πακέτο που θα υπακούσει .Αφού κάνει ταυτοποιήση της διεύθυνσης της θύρα και του είδους του πακέτου τότε ελέγχει όλο το socket και αφού ενώσει τα πακέτα που έχει δεχθεί με την ταυτοποίηση που έκανε κλείνει το socket αφού προηγουμένως έχει φτιάξει έναν χάρτη με τους δέκτες που έχουν κάνει αίτηση για να λάβουν τα πακέτα και τους έχει βάλει σε μια κοινή διεύθυνση (multicast join group). Σε κάθε περίπτωση που δεν έχει τερματιστεί η σύνδεση και εμείς προσπαθήσουμε να στήλουμε αίτηση στην ίδια διεύθυνση τότε μας εμφανίζεται ότι η διεύθυνση χρησιμοποιείται (bind) , αν πάλλι δεν έχουμε δηλώσει διεύθυνση μας εμφανίζει μήνυμα να δηλώσουμε(πάρουμε) μια διεύθυνση για να ολοκληρωθεί η επικοινωνία.

Επειδή θέλουμε να στήλουμε μια εικόνα 256X256X3 μέσω του δικτύου αλλα επειδή μας περιορίζει το μέγεθος του udp πακέτου σε 65500 bytes θα αναγκαστούμε να λάβουμε την εικόνα σε 4 πακέτα των 49152 bytes αφού το μέγεθος της εικόνας είναι 196608. Αρχικά βάζουμε μια λούπα επανάληψης αγνώστου οριακού χρόνου επειδή θέλουμε να στήλουμε πολλές εικόνες έτσι ώστε αφού λάμβάνουμε καθε φορά μια εικόνα να μηδενίζεται ο buffer για να εκχωρίσουμε την επόμενη ,αυτή η λούπα τρέχει απο την στιγμή που λαμβάνουμε τους 4 buffer μέχρι την εμφάνιση της εικόνας .

Όπως και στον αποστολέα που είχαμε την συνθήκη sendto τώρα έχουμε την recvfrom. Όπου αρχικοποιείται σε μια μεταβλητή που ελέγχει το μέγεθος του πακέτου που λαμβάνουμε . Στη συνθήκη αυτή έχουμε ως μεταβλητές το μέγεθος του buffer ,μια μεταβλητή που συνδέεται με το socket ,το όνομα του buffer και μια δομή

που ελέγχει μέσω των δεικτών του απο πού ερχεται το πακέτο μέσω του socket . Αφού επαληθευθούν ο μεταβλητές αυτές τότε εμφανίζεται μήνυμα οτι ο buffer λήφθηκε σωστά.Αυτή η διαδικασία επαναλαμβάνεται 4 φορές όσοι είναι και οι buffer. Αφού λάβουμε και τους 4 buffer θα πρέπει να τους ενώσουμε έτσι ώστε να επιξεργαστούμε την εικόνα ολόκληρη . Για αυτό δημιουργούμε 4 λούπες επαναληψης for οι οποίες επαναλαμβάνονται για όσο είνια και το μέγεθος του κάθε buffer δηλαδή 49152 και τις αρχικοποιούμε σε έναν πινακα tmpim που περιέχει το συνολικό μέγεθος της εικόνας δηλαδή 196608. Αφού αρχικοποιείσουμε τον πρώτο buffer στον πίνακα tmpim στον δεύτερο buffer προσθέτουμε τον προήγούμενο και αυτό επαναλαμβανεται μέχρι ο tmpim να επαληθεύει το μέγεθος του με αυτό της συνολικής εικόνας, τότε μας εμφανίζεται ένα μήνυμα ότι η εικόνα δημιουργήθηκε .

```
for(;;)

{

   size_t len1;

   len1 = recvfrom( s, buf1, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

   printf( "Received BUF1\n", buf1 );

   size_t len2;

   len2 = recvfrom( s, buf2, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

   printf( "Received BUF2\n", buf2 );

   size_t len3;

   len3 = recvfrom( s, buf3, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

   printf( "Received BUF3\n", buf3 );

   size_t len4;

   len4 = recvfrom( s, buf4, 49152, 0,(struct sockaddr *)&from,&fromlen) ;
```

```
   printf( "Received BUF4\n", buf4 );

  printf("ENSWMATWSI TWN 4 BUFFER SE SE ENAN PINAKA TMPIM\n");

for(q=0; q<49152; q++)

                {

        tmpim[q] = buf1[q];

                }

for(w=0; w<49152; w++)

                {

        tmpim[49152+w] = buf2[w];

                }

for(e=0; e<49152; e++)

        {

        tmpim[98304+e] = buf3[e];

        }

for(r=0; r<49152; r++)

         {

        tmpim[147456+r] = buf4[r];

        }

printf("o pinakas  dimiourgithike\n",tmpim[i]);
```

Όπως ξέρουμε η ποιότητα των υπηρεσιών που προσφέρει το ipv6 είναι καλύτερο απο ότι το ipv4 και με αυτο τον τρώπο λάθος εικόνες δεν εμφανίζονται έτσι για να

προσομοιώσουμε με κατάσταση τέτοια παίρνουμε μια τυχαία τιμή μέσω μιας συνάρτησης rand(). Η συνάρτηση αυτή παίρνει τιμές απο το 1 μεχρι το 10 και όταν παίρνει τιμή 3 παίρνει τον πίνακα tmpim και τον αρχικοποιεί λανθασμένα σε έναν πίνακα img[256][256][3] που περιέχει τα χρωματα και έτσι η εικονα μας εμφανίζεται λάθος, αντιθέτως όταν δεν παίρνει τιμή ίση με 3 τότε ο πίνακας αρχικοποιείται σωστά στον πινακα img. Όπου οι πίνακες αυτοί εκφολεύονται σε έναν δισδιάστατο πίνακα 256 επι 256 όσος είναι και ο πίνακας img και μέσω της λούπας αυτής επαληθεύεται η εκχόριση του tmpim στον πίνακα img 256X256 φορές. Κάθε χρώμα απο τα βασικα του bmp format κοκκινο πράσινο και μπλέ εκχορούνται σε ξεχωριστό πίνακα img.

```
r=rand()%10+1;

printf("%d\n",r);

if(r!=3)

{

for(i=0; i<256; i++)

{

for( j=0; j<256; j++)

{

  img[i][j][2]=tmpim[(j*256+256-i-1)*3+2];

  img[i][j][1]=tmpim[(j*256+256-i-1)*3+1];

  img[i][j][0]=tmpim[(j*256+256-i-1)*3+0];

}

printf("\n");

}
```

```
}
else if(r==3)
{
for(i=0; i<256; i++)
{
  for( j=0; j<256; j++)
{
   img[i][j][2]=tmpim[(j*256+256-i*3+1)*3+2];
   img[i][j][1]=tmpim[(j*256+256-i*3+1)*3+1];
   img[i][j][0]=tmpim[(j*256+256-i*3+1)*3+0];
}
printf("\n");
}
}
```

Όπως βλέπουμε ουσιαστικά αφού έχουμε εισάγει την εικόνα στον πίνακα tmpim τώρα σχεδιάζουμε και την διάταξη της εικόνας οριζόντια και κάθετα έτσι ώστε να εισάγουμε τα τρία χρώματα.

Μέσω μιάς επαναλαμβανόμενης λούπας switch οι οποία παίρνει εντολές απο τον χρήστη ρωτάμε τον χρήστη αν θέλει να συνεχίσει την λήψη ή να την τερματίσει. Αφού συνεχιστεί η λήψη εμφανιζεται μήνυμα οτι συνεχίζεται η διαδικασία και μέσω μια άλλης switch θα πρέπει να λαμβάνουμε την εικόνα αφού δώσουμε επιβεβαίωση με το όνομα της . Όταν γίνει αυτό μας λέει πια εικόνα λάμβάνουμε. Αν σε κάθε

περίπτωση έχουμε πάρει τιμή ίση με 3 μέσω της συνάρτησης τυχαίων τιμών rand() τότε εμφανίζεται μήνυμα ότι λάμβάνουμε λάθος την εικόνα .

```
 char ch,choice;

printf("leipsi eikwnwn\n");

printf("DEXESTE TA BMP ARXEIA? PLIKTROLOGISTE Y OR N \n",'Y','N');

scanf("%s",&ch);

switch(ch)

{

case 'Y':

{

printf("perimenete oso ginetai i metavivasi\n");

break;

}

case 'N':

{

exit(1);

}

}

printf("AN THES NA LAVEIS TIS AKOLOYTHES EIKONES PATA APO A - J KATA SEIRA PARALAVIS :\n");

scanf("%s",&choice);
```

```
switch(choice)

{

case 'A':

{

f=fopen("Anemos.bmp","wb");

printf("lamvanetai  o Anemos\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");

}

break;

}

case 'B':

{

f=fopen("Anemos1.bmp","wb");

printf("lamvanetai  o Anemos1\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
```

```
}
break;
}
case 'C':
{
f=fopen("Anemos2.bmp","wb");
printf("lamvanetai  o Anemos2\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'D':
{
f=fopen("Anemos3.bmp","wb");
printf("lamvanetai  o Anemos3\n");
if(r==3)
{
```

```
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");

}

break;

}

case 'E':

{

f=fopen("Anemos4.bmp","wb");

printf("lamvanetai  o Anemos4\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");

}

break;

}

case 'F':

{

f=fopen("Anemos5.bmp","wb");

printf("lamvanetai  o Anemos5\n");

if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'G':
{
f=fopen("Anemos6.bmp","wb");
printf("lamvanetai  o Anemos6\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'H':
{
f=fopen("Anemos7.bmp","wb");
printf("lamvanetai  o Anemos7\n");
```

```
if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");

}

break;

}

case 'I':

{

f=fopen("Anemos8.bmp","wb");

printf("lamvanetai  o Anemos8\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS POIOTITAS\n");

}

break;

}

case 'J':

{

f=fopen("Anemos9.bmp","wb");
```

```
printf("lamvanetai  o Anemos8\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS POIOTITAS\n");

}

break;

}

}
```

Αφού λάβουμε τις εικόνες και αρχικά τις ενσωματώσουμε με τα χρωματά τους στους πίνακες θα φτάσουμε στο επίπεδο όπου θα πρέπει να βάλουμε αυτά τα στοιχεία σε μια σειρά μέσω της κεφαλιδας(header) που περέχει τα στοιχεία για τις διαστάσεις της εικόνας την συμπίεση το μεγεθος της κεφαλίδας τις διαστάσεις των εικονοστηχείων μέσα στην εικόνα(pixelpermetrix) και τον τρόπο που αναλύουμε τα χρώματα μέσω του αριθμού των bits ανάλυσης δηλαδή 4,8,16,24 και 32 bits τα οποία με την σειρά τους έχουν και ένα σύνολο χρωμάτων τα οποία μπορούν να εμφανίσουν. Το σύνολο της κεφαλίδας σε bytes είναι 54 , 40 για το infoheader που περιέχει όλα τα στοιχεία της κεφαλιδας και 14 για την κεφαλίδα αρχείου(DIBheader).

```
//write header//

 //FILE HEADER//

 //"BM"//

a= 'B';fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 'M'; fwrite(&a,sizeof(char),1,f); // write a, one single byte

//// Size   (256*256*4+54)=262198 = 4*(256^2)+0*256+54
```

```
a= 0x36;        fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0x00;        fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0x04;        fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0;           fwrite(&a,sizeof(char),1,f); // write a, one single byte

//Rsvd

a= 0;    fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0;    fwrite(&a,sizeof(char),1,f); // write a, one single byte

//Rsvd

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

// Bytesbeforepixels

a=54;   fwrite(&a,1,1,f); // write a, one single byte

a=0;     fwrite(&a,1,1,f); // write a, one single byte

a=0;     fwrite(&a,1,1,f); // write a, one single byte

a=0;     fwrite(&a,1,1,f); // write a, one single byte

//BITMAPINFO

//BITMAPINFOHEADER

//Size of header

a= 40;  fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte
```

```
a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//Width

a= (char)0;fwrite(&a,1,1,f); // write a, one single byte

a= 1;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//Height

a= (char)0;fwrite(&a,1,1,f); // write a, one single byte

a= 1;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//Planes

a= 1;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//BitCount

a= 24;  fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//Compression

a= 0;    fwrite(&a,1,1,f); // write a, one single byte
```

```
a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//Image Size (since we have no compression, we can use 0)

a= 0x0;          fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0x00;fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0x04;fwrite(&a,sizeof(char),1,f); // write a, one single byte

a= 0;    fwrite(&a,sizeof(char),1,f); // write a, one single byte

//pixelspermeterX

a= 0xC4;         fwrite(&a,1,1,f); // write a, one single byte

a= 0xE;          fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//clrused

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

//ClrImportant

a= 0;    fwrite(&a,1,1,f); // write a, one single byte
```

```
a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;    fwrite(&a,1,1,f); // write a, one single byte
```

Αφού περιγράψαμε πως είναι η κεφαλιδα(header) όταν φτάσει στο σημειο αυτο και η εικόνα θα έχει σχεδιαστεί πλήρως θα πρέπει με κάπιο τρόπο να την εμφανίσουμε , για αυτο αρχικοποιούμε τον πίνακα img για το κάθε χρώμα σε τρείς μεταβλητές που περιγράψουν το καθε χρώμα μέσα σε έναν δισδιάστατο πινακα που έχει το μέγεθος του οριζόντιου και κάθετου μεγέθους της εικόνας δηλαδή 256X256 και μια συνθήκη που ελέγχει αν οι πίνακες έχουν τρέξει τον δισδιάστατο πίνακα . Η εντολή η οποία μας δινει την δυνατότητα να εμφανίσουμε κάτι το οποία δεν εχει σχέση με αλφαριθμυτική τιμή είναι η εντολή write που στην περίπτωση μας επειδή εμφανίζει αρχείο είναι fwrite.

```
for (i=0;i<256;i++)

{

for (j=0; j<256;j++)

{

if ((i<256)&&(j<256))

{

cr= img[i][j][2] ; // red

cg= img[i][j][1] ; // green

cb= img[i][j][0] ; // blue

}

else
```

```
{

cr=0;

cg=0;

cb=0;

}

fwrite(&cr,1,1,f); //r

fwrite(&cg,1,1,f); //g

fwrite(&cb,1,1,f); //b

}

}
```

Όπως βλέπου στην περίπτωση που οι μεταβλητές στις οποίες αρχικοποιούνται οι πίνακες img με το κάθε χρώμα μηδενίζονται αν δεν πάρουν κάποια τιμή .Τελικά αναλόγως με το σύνολο των εικόνων που λάβαμε λάθος έχουμε μια πράξη που διαιρεί το σύνολο των λανθασμένων εικόνων με αυτό του συνόλου των εικόνων και μας εμφανίζει το ποσοστό λάθους.

```
printf("... IMAGES received...\n");

int width;

float A;

printf("An exeis lavei lathos eikones dwse to sinolo twn lanthasmenwn eikonwn gia na vroume to pososto lathous:\n");

scanf("%d",&width);

A=(float)width/10.0*100.0;
```

```
printf("To pososto einai %f\n",A);

fclose(f);

}

}
```

Για περαιτέρω αναλύσεις στις συνθήκες επανάληψης και άλλους κανόνες της γλώσσας C θα βρείτε στο ΠΑΡΑΤΗΜΑ Β .

4.4 Κώδικας αποστολέα

```
#include <sys/types.h>   /* for type definitions */

#include <sys/socket.h>  /* for socket API function calls */

#include <netinet/in.h>  /* for address structs */

#include <arpa/inet.h>   /* for sockaddr_in */

#include <stdio.h>       /* for printf() */

#include <stdlib.h>      /* for atoi() */

#include <string.h>      /* for strlen() */

#include <unistd.h>      /* for close() */

#include <errno.h>

#include <net/if.h>

#include <netdb.h>

#include <stdint.h>

#include <stdbool.h>

#include <assert.h>
```

```
int main (int argc,char **argv)

{

int error;

int s;

int q,w,e,r;

unsigned char buf1 [49152];

unsigned char buf2 [49152];

unsigned char buf3 [49152];

unsigned char buf4 [49152];

size_t len;

FILE * fp;

unsigned long cnt,im_row,im_col;

unsigned char tc;

unsigned int w1,w2,width,height,h1,h2,offset,bpp,j;

unsigned int padd,i,l;

unsigned char image[256][256][3];

unsigned char tmpim[196608];

struct addrinfo hint, *res;

struct addrinfo *multi;

memset( &hint, 0, sizeof( hint ) );

hint.ai_family = AF_INET6;
```

```
hint.ai_socktype = SOCK_DGRAM;

hint.ai_protocol = 0;

int count=0;

error = getaddrinfo( "ff02::1", NULL, &hint, &res );

if( error != 0 ) {

perror( "getaddrinfo" );

}

struct sockaddr_in6 * addr = (struct sockaddr_in6*)res->ai_addr;

addr->sin6_port = htons( 7890 );

addr->sin6_scope_id = 2; // 2 happens to be the interface ID

s = socket( AF_INET6, SOCK_DGRAM, 0 );

char ch;

printf("STEILE TIS EIKONES POU THELEIS\n");

printf("pata A gia na steileis tin prwti eikona ANEMOS\n",ch);

printf("pata B gia na steileis tin deuteri eikona ANEMOS1\n",ch);

printf("pata C gia na steileis tin triti eikona ANEMOS2\n",ch);

printf("pata D gia na steileis tin tetarti eikona ANEMOS3\n",ch);

printf("pata E gia na steileis tin pempti eikona ANEMOS4\n",ch);

printf("pata F gia na steileis tin ekti eikona ANEMOS5\n",ch);

printf("pata G gia na steileis tin evdomi eikona ANEMOS6\n",ch);

printf("pata H gia na steileis tin ogdoi eikona ANEMOS7\n",ch);
```

```
printf("pata I gia na steileis tin enati eikona ANEMOS8\n",ch);
printf("pata J gia na steileis tin dekati eikona ANEMOS9\n",ch);
scanf("%s",&ch);
do
{
switch(ch)
{
case 'A':
{
if (!(fp=fopen("Anemos.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos.bmp");
}
count++;
printf("stelnetai o Anemos\n");
break;
}
case 'B':
{
if (!(fp=fopen("Anemos1.bmp","rb")))
{
```

```
printf("Unable to load bin file: %s...\n","Anemos1.bmp");
}
count++;
printf("stelnetai o Anemos1\n");
break;
}
case 'C':
{
if (!(fp=fopen("Anemos2.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos2.bmp");
}
count++;
printf("stelnetai o Anemos2\n");
break;
}
case 'D':
{
if (!(fp=fopen("Anemos3.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos3.bmp");
```

```
}
count++;
printf("stelnetai o Anemos3\n");
break;
}
case 'E':
{
if (!(fp=fopen("Anemos4.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos4.bmp");
}
count++;
printf("stelnetai o Anemos4\n");
break;
}
case 'F':
{
if (!(fp=fopen("Anemos5.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos5.bmp");
}
```

```
count++;
printf("stelnetai o Anemos5\n");
break;
}
case 'G':
{
if (!(fp=fopen("Anemos6.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos6.bmp");
}
count++;
printf("stelnetai o Anemos6\n");
break;
}
case 'H':
{
if (!(fp=fopen("Anemos7.bmp","rb")))
{
printf("Unable to load bin file: %s...\n","Anemos7.bmp");
}
count++;
```

```
printf("stelnetai o Anemos7\n");

break;

}

case 'I':

{

if (!(fp=fopen("Anemos8.bmp","rb")))

{

printf("Unable to load bin file: %s...\n","Anemos8.bmp");

}

count++;

printf("stelnetai o Anemos8\n");

break;

}

case 'J':

{

if (!(fp=fopen("Anemos9.bmp","rb")))

{

printf("Unable to load bin file: %s...\n","Anemos9.bmp");

}

count++;

printf("stelnetai o Anemos9\n");
```

```
break;
}
}
cnt = 0;
im_row=0;
im_col=0;
while (!feof(fp)) {
tc = fgetc(fp) & 0x0ff;
if (cnt==10) offset=tc;
if (cnt==18) w1=tc;
if (cnt==19) w2=tc;
if (cnt==22) h1=tc;
if (cnt==23) h2=tc;
if (cnt==28) bpp=tc;
if ((cnt==30)&&(tc>0)) printf("ERROR!! Cannot Handle BMP files with Compression!!!\n");
 if (cnt==53) break;
cnt++;
}
width  = w1+w2*256;
height = h1+h2*256;
int  BMPRES_X = height;
```

```
int  BMPRES_Y = width;

padd = 0;

printf("W:%d, H:%d Image Data Starts at:%d with %d bits/pixel \n",width,height,offset,bpp);

for (im_row=0;im_row<height;im_row++)

{

 for (im_col=0;im_col<width;im_col++)

{

tc = fgetc(fp) & 0x0ff;

tmpim[(im_col*height+height-im_row-1)*3+2] = tc;

 if ((im_col%32==0)&&(im_row%32==0)) printf("%02u",tc/10);

tc = fgetc(fp) & 0x0ff;

tmpim[(im_col*height+height-im_row-1)*3+1] = tc;

 if ((im_col%32==0)&&(im_row%32==0)) printf("%02u",tc/10);

 tc = fgetc(fp) & 0x0ff;

tmpim[(im_col*height+height-im_row-1)*3+0] = tc;

if ((im_col%32==0)&&(im_row%32==0)) printf("%02u-",tc/10);

}

 if (im_row%32==0) printf("\n");

for (i=0;i<padd;i++)

{

 tc = fgetc(fp) & 0x0ff;
```

```
}

}

fclose(fp);

for (i=0; i<height; i++) {

for (j=0; j<width; j++) {

image[i][j][2] = (int)(tmpim[(j*height+height-i-1)*3+2]); // blue

image[i][j][1] = (int)(tmpim[(j*height+height-i-1)*3+1]); // green

image[i][j][0] = (int)(tmpim[(j*height+height-i-1)*3+0]);// red

} // end of line

}

int tmp; printf("\n ...IMAGE READ...\n");

scanf("%d",&tmp);

q=0;

for(i=1; i<=64; i++)

{

for(j=1; j<=256; j++)

{

buf1[q] = image[i][j][2];

q++;

 buf1[q] = image[i][j][1];

q++;
```

```
 buf1[q] = image[i][j][0];
q++;
}
}
printf("1st BUFFER OK...\n");
scanf("%d",&tmp);
w=0;
for(i=65; i<=128; i++)
{
for(j=1; j<=256; j++)
{
buf2[w] = image[i][j][2];
w++;
buf2[w] = image[i][j][1];
w++;
buf2[w] = image[i][j][0];
w++;
}
}
printf("2nd BUFFER OK...\n");
scanf("%d",&tmp);
```

```
e=0;
for(i=129; i<=192; i++)
{
for(j=1; j<=256; j++)
{
buf3[e] = image[i][j][2];
e++;
buf3[e] = image[i][j][1];
e++;
buf3[e] = image[i][j][0];
e++;
}
}
printf("3rd BUFFER OK...\n");
scanf("%d",&tmp);
r=0;
for(i=193; i<=256; i++)
{
for(j=1; j<=256; j++)
{
buf4[r] = image[i][j][2];
```

```
r++;
buf4[r] = image[i][j][1];
r++;
buf4[r] = image[i][j][0];
r++;
}
}
printf("4rd BUFFER OK...\n");
if( sendto( s, buf1, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
printf( "Error sending buf1\n" );
printf(" afou dn stalthike o protos buffer tote diakoptete i apostoli\n");
exit(1);
} else printf("Buf1 OK! \n",buf1);
if( sendto( s, buf2, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
printf( "Error sending buf2\n" );
} else printf("Buf2 OK!\n");
if( sendto( s, buf3, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
printf( "Error sending buf3\n" );
} else printf("Buf3 OK!\n");
if( sendto( s, buf4, 49152, 0, res->ai_addr, res->ai_addrlen ) !=49152) {
printf( "Error sending buf4\n" );
```

```
}else printf("Buf4 OK!\n");

printf("Mexri twra exoume stilei %d eikones\n",count);

printf("pata ena gramma  apo A-J gia tin apostoli twn eikonwn i pata X gia to kleisimo:\n");

scanf("%s",&ch);

}while(ch<='J');

exit(1);

}
```

4.5 Κώδικας δέκτη

```
#include <sys/types.h>   /* for type definitions */

#include <sys/socket.h>  /* for socket API function calls */

#include <netinet/in.h>  /* for address structs */

#include <arpa/inet.h>   /* for sockaddr_in */

#include <stdio.h>       /* for printf() */

#include <stdlib.h>      /* for atoi() */

#include <string.h>      /* for strlen() */

#include <unistd.h>      /* for close() */

#include <errno.h>

#include <net/if.h>

#include <netdb.h>

#include <stdint.h>

#include <stdbool.h>
```

```
#include <assert.h>

int main(int argc, char** argv)

{

int error;

int s;

struct addrinfo hint, *res;

struct addrinfo *multi;

struct addrinfo from;

int fromlen;

int x,q,w,e,r,a,y;

FILE* f;

unsigned char buf1[49152];

unsigned char buf2[49152];

unsigned char buf3[49152];

unsigned char buf4[49152];

unsigned char tmpim[196608];

unsigned int i,j;

memset( &hint, 0, sizeof( hint ) );

hint.ai_family = AF_INET6;

hint.ai_socktype = SOCK_DGRAM;

error = getaddrinfo( "ff02::1", NULL, &hint, &res );
```

```
if( error != 0 ) {

perror( "getaddrinfo" );

exit( 1 );

}

struct sockaddr_in6 * addr = (struct sockaddr_in6*)res->ai_addr;

addr->sin6_addr = in6addr_any;

addr->sin6_port = htons( 7890 );

s = socket( AF_INET6, SOCK_DGRAM, 0 );

if( bind( s, (struct sockaddr*) addr, res->ai_addrlen ) != 0 ) {

close( s );

perror( "bind" );

exit( 1 );

}

if( getaddrinfo( "ff02::1", NULL, &hint, &multi ) != 0 ) {

close( s );

perror( "getaddrinfo" );

}

struct ipv6_mreq mreq;

memset( &mreq, 0, sizeof(mreq) );

memcpy( &mreq.ipv6mr_multiaddr, &((struct sockaddr_in6 *) multi->ai_addr)->sin6_addr,
 sizeof(mreq.ipv6mr_multiaddr) );

mreq.ipv6mr_interface = 2; // 2 happens to be the interface ID; I've tried other values here
```

```
freeaddrinfo( multi );

if( setsockopt( s, IPPROTO_IPV6, IPV6_JOIN_GROUP, &mreq, sizeof(mreq) ) != 0 )

{

close( s );

perror( "IPV6_JOIN_GROUP" );

}

for(;;)

{

size_t len1;

len1 = recvfrom( s, buf1, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

printf( "Received BUF1\n", buf1 );

size_t len2;

len2 = recvfrom( s, buf2, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

printf( "Received BUF2\n", buf2 );

size_t len3;

len3 = recvfrom( s, buf3, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

printf( "Received BUF3\n", buf3 );

size_t len4;

len4 = recvfrom( s, buf4, 49152, 0,(struct sockaddr *)&from,&fromlen) ;

printf( "Received BUF4\n", buf4 );

printf("ENSWMATWSI TWN 4 BUFFER SE SE ENAN PINAKA TMPIM\n");
```

```
for(q=0; q<49152; q++)
 {
tmpim[q] = buf1[q];
 }
 for(w=0; w<49152; w++)
 {
tmpim[49152+w] = buf2[w];
}
for(e=0; e<49152; e++)
{
tmpim[98304+e] = buf3[e];
}
for(r=0; r<49152; r++)
{
tmpim[147456+r] = buf4[r];
}
printf("o pinakas  dimiourgithike\n",tmpim[i]);
unsigned char img[256][256][3];
unsigned char padd=0;
int r,cnt=0;
r=rand()%10+1;
```

```
printf("%d\n",r);
if(r!=3)
{
for(i=0; i<256; i++)
{
for( j=0; j<256; j++)
{
img[i][j][2]=tmpim[(j*256+256-i-1)*3+2];
img[i][j][1]=tmpim[(j*256+256-i-1)*3+1];
img[i][j][0]=tmpim[(j*256+256-i-1)*3+0];
}
printf("\n");
}
}
else if(r==3)
{
for(i=0; i<256; i++)
{
for( j=0; j<256; j++)
{
img[i][j][2]=tmpim[(j*256+256-i*3+1)*3+2];
```

```
img[i][j][1]=tmpim[(j*256+256-i*3+1)*3+1];

img[i][j][0]=tmpim[(j*256+256-i*3+1)*3+0];

}

printf("\n");

}

}

char ch,choice;

printf("leipsi eikwnwn\n");

printf("DEXESTE TA BMP ARXEIA? PLIKTROLOGISTE Y OR N \n",'Y','N');

scanf("%s",&ch);

switch(ch)

{

case 'Y':

{

printf("perimenete oso ginetai i metavivasi\n");

break;

}

case 'N':

{

exit(1);

}
```

```
}

printf("AN THES NA LAVEIS TIS AKOLOYTHES EIKONES PATA APO A - J KATA
 SEIRA PARALAVIS :\n");

scanf("%s",&choice);

switch(choice)

{

case 'A':

{

f=fopen("Anemos.bmp","wb");

printf("lamvanetai  o Anemos\n");

if(r==3)

{

printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS
 PIOTITAS\n");

}

break;

}

case 'B':

{

f=fopen("Anemos1.bmp","wb");

printf("lamvanetai  o Anemos1\n");

if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'C':
{
f=fopen("Anemos2.bmp","wb");
printf("lamvanetai  o Anemos2\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'D':
{
f=fopen("Anemos3.bmp","wb");
printf("lamvanetai  o Anemos3\n");
if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'E':
{
f=fopen("Anemos4.bmp","wb");
printf("lamvanetai  o Anemos4\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'F':
{
f=fopen("Anemos5.bmp","wb");
printf("lamvanetai  o Anemos5\n");
if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'G':
{
f=fopen("Anemos6.bmp","wb");
printf("lamvanetai  o Anemos6\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'H':
{
f=fopen("Anemos7.bmp","wb");
printf("lamvanetai  o Anemos7\n");
if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS PIOTITAS\n");
}
break;
}
case 'I':
{
f=fopen("Anemos8.bmp","wb");
printf("lamvanetai  o Anemos8\n");
if(r==3)
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS POIOTITAS\n");
}
break;
}
case 'J':
{
f=fopen("Anemos9.bmp","wb");
printf("lamvanetai  o Anemos8\n");
if(r==3)
```

```
{
printf("H AKOLOYTHI EIKONA THA LIFTHEI ME LATHOI LOGO KAKIS POIOTITAS\n");
}
break;
}
}
//write header//
//FILE HEADER//
//"BM"//
 a= 'B';fwrite(&a,sizeof(char),1,f); // write a, one single byte
 a= 'M';  fwrite(&a,sizeof(char),1,f); // write a, one single byte
//// Size   (256*256*4+54)=262198 = 4*(256^2)+0*256+54
 a= 0x36;          fwrite(&a,sizeof(char),1,f); // write a, one single byte
 a= 0x00;          fwrite(&a,sizeof(char),1,f); // write a, one single byte
 a= 0x04;          fwrite(&a,sizeof(char),1,f); // write a, one single byte
 a= 0;             fwrite(&a,sizeof(char),1,f); // write a, one single byte
 //Rsvd
 a= 0;     fwrite(&a,sizeof(char),1,f); // write a, one single byte
 a= 0;     fwrite(&a,sizeof(char),1,f); // write a, one single byte
 //Rsvd
 a= 0;     fwrite(&a,1,1,f); // write a, one single byte
```

```
 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 // Bytesbeforepixels

 a=54;    fwrite(&a,1,1,f); // write a, one single byte

 a=0;      fwrite(&a,1,1,f); // write a, one single byte

 a=0;      fwrite(&a,1,1,f); // write a, one single byte

 a=0;      fwrite(&a,1,1,f); // write a, one single byte

//BITMAPINFO

//BITMAPINFOHEADER

//Size of header

a= 40;    fwrite(&a,1,1,f); // write a, one single byte

a= 0;     fwrite(&a,1,1,f); // write a, one single byte

a= 0;     fwrite(&a,1,1,f); // write a, one single byte

a= 0;     fwrite(&a,1,1,f); // write a, one single byte

//Width

a= (char)0;fwrite(&a,1,1,f); // write a, one single byte

a= 1;     fwrite(&a,1,1,f); // write a, one single byte

a= 0;     fwrite(&a,1,1,f); // write a, one single byte

a= 0;     fwrite(&a,1,1,f); // write a, one single byte

//Height

a= (char)0;fwrite(&a,1,1,f); // write a, one single byte

a= 1;     fwrite(&a,1,1,f); // write a, one single byte
```

```
a= 0;      fwrite(&a,1,1,f); // write a, one single byte

a= 0;      fwrite(&a,1,1,f); // write a, one single byte

//Planes

a= 1;      fwrite(&a,1,1,f); // write a, one single byte

a= 0;      fwrite(&a,1,1,f); // write a, one single byte

//BitCount

 a= 24;   fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 //Compression

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 //Image Size (since we have no compression, we can use 0)

 a= 0x0; fwrite(&a,sizeof(char),1,f); // write a, one single byte

 a= 0x00;fwrite(&a,sizeof(char),1,f); // write a, one single byte

 a= 0x04;fwrite(&a,sizeof(char),1,f); // write a, one single byte

 a= 0;     fwrite(&a,sizeof(char),1,f); // write a, one single byte

 //pixelspermeterX

 a= 0xC4;          fwrite(&a,1,1,f); // write a, one single byte

 a= 0xE; fwrite(&a,1,1,f); // write a, one single byte
```

```
 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 //Clrused

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 //ClrImportant

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

 a= 0;     fwrite(&a,1,1,f); // write a, one single byte

int cr, cb, cg;

 for (i=0;i<256;i++)

 {

for (j=0; j<256;j++)

{

 if ((i<256)&&(j<256))

 {

cr= img[i][j][2] ; // red

cg= img[i][j][1] ; // green
```

```
cb= img[i][j][0] ; // blue
 }
 else
 {
 cr=0;
 cg=0;
 cb=0;
 }
 fwrite(&cr,1,1,f); //r
 fwrite(&cg,1,1,f); //g
 fwrite(&cb,1,1,f); //b
 }
 }
printf("... IMAGES received...\n");
int width;
float A;
printf("An exeis lavei lathos eikones dwse to sinolo twn lanthasmenwn eikonwn gia na
 vroume to pososto lathous:\n");
scanf("%d",&width);
A=(float)width/10.0*100.0;
printf("To pososto einai %f\n",A);
fclose(f);}}
```

Παράρτημα Α: Βιβλιογραφία και πηγές απο το διαδίκτυο

ΒΙΒΛΙΑ

C FOR SCIENTISTS ENGINEERS,RICHARD JOHNSONBAUGH AND MARTIN KALIN

C:STEP BY STEP ,M.GIOURDAS

IPV6 NETWORK PROGRAMMING ,JUN-ICHIRO ITOJUN HAGINO

Η ΓΛΩΣΣΑ C ΣΕ ΒΑΘΟΣ ,ΝΙΚΟΣ Μ.ΧΑΤΖΗΓΙΑΝΝΗΣ

C ΓΙΑ ΜΗΧΑΝΙΚΟΥΣ, H.H.TAN T/B/D'ORAZIO

CISCO MAGAZINE MULTICASTING IPV6 AND NETWORK

ΙΣΤΟΤΟΠΟΙ

http://en.wikipedia.org/wiki/Multicast

http://en.wikipedia.org/wiki/Multicast_Source_Discovery_Protocol

http://en.wikipedia.org/wiki/IP_multicast

http://en.wikipedia.org/wiki/Multicast_address

http://technet.microsoft.com/en-us/library/cc781068(v=ws.10).aspx

http://blog.ine.com/2009/12/16/ipv6-multicast-addressing/

www.ipv6-es.com/.../davi

http://en.wikipedia.org/wiki/IPv6

http://www.cisco.com/en/US/docs/ios/ipv6/configuration/guide/ip6-multicast.html

www.6diss.org/.../***multica***

Παράρτημα Β: Κανόνες της γλώσσας C

Τύποι δεδομένων

Η C έχει πέντε βασικούς τύπους δεδομένων:

char(character),

int(integer),

float(floating point),

double(double floating point),

void(no value).

Όλοι οι άλλοι τύποι της C βασίζονται σ' αυτούς. Όλοι οι βασικοί τύποι εκτός από τον τύπο void μπορεί ν' αλλάξουν γράφοντας πριν από τον τύπο τον κατάλληλο μετασχηματισμό. Οι μετασχηματισμοί αυτοί είναι οι**: signed, unsigned, long, και short.**Το μέγεθος και τα διαστήματα τιμών των τύπων της C εξαρτάται από τον επεξεργαστή. Στον πίνακα δίνουμε τους τύπους δεδομένων όπως ορίζονται από το πρότυπο ANSI.

Δηλώσεις μεταβλητών

Τα αναγνωριστικά στη C μπορούν να έχουν όσους χαρακτήρες θέλουμε. Αν το αναγνωριστικό είναι εξωτερικό όνομα (όνομα συνάρτησης η καθολική μεταβλητή) τότε μόνο οι έξι πρώτοι χαρακτήρες είναι σημαντικοί διαφορετικά για εσωτερικά ονόματα οι πρώτοι 31 χαρακτήρες είναι σημαντικοί.**Τα κεφαλαία γράμματα στην C είναι διαφορετικά από τα μικρά.**

Η δήλωση μιας μεταβλητής έχει την γενική μορφή:

<τύπος> <λίστα μεταβλητών>

int i=0,j;

char q='?';

short int si;

float f,g;

Δηλώσεις μεταβλητών κάνουμε μέσα σε συναρτήσεις (local variables, automatic), στις παραμέτρους μιας συνάρτησης (formal parameters) και έξω απ' όλες τις συναρτήσεις (global variables). Μπορούμε επίσης να δηλώσουμε μία τοπική μεταβλητή μέσα σε μία ενότητα π.χ.

if (συνθήκη)

{

char x[30];

...

...

}

Στη περίπτωση αυτή η εμβέλεια της μεταβλητής είναι η ενότητα στην οποία είναι δηλωμένη. Έτσι αποφεύγουμε πλάγια αποτελέσματα και έχουμε οικονομία χώρου. Καθολικές μεταβλητές έχουν εμβέλεια σε ολόκληρο το πρόγραμμα και δηλώνονται στην αρχή του κώδικα έξω απ' όλες τις συναρτήσεις. Όταν μια καθολική και μία τοπική μεταβλητή έχουν το ίδιο όνομα τότε μέσα στην εμβέλεια της τοπικής μεταβλητής αναφερόμαστε πάντα στην τοπική. Μεταβλητές που είναι παράμετροι συναρτήσεων συμπεριφέρονται ως τοπικές μεταβλητές της συνάρτησης.

Σταθερές εισάγονται με την προκαθορισμένη λέξη **const** π.χ.

const int i=1;

const char q='?';

Πτητικές (volatile) μεταβλητές πληροφορούν τον μεταγλωττιστή ότι η τιμή τους μπορεί ν' αλλάξει χωρίς αυτό να δηλώνεται σαφώς στο πρόγραμμα.

Τρόποι αποθήκευσης μεταβλητών

Υπάρχουν τέσσερις τρόποι να πληροφορήσουμε τον μεταγλωττιστή πως ν' αποθηκευτεί μια μεταβλητή.

extern

Επειδή η C υποστηρίζει ξεχωριστή μεταγλώττιση των διαφόρων ενοτήτων ενός μεγάλου προγράμματος θα πρέπει να υπάρχει κάποιος τρόπος που να πληροφορεί τον μεταγλωττιστή ότι ορισμένες μεταβλητές είναι ορισμένες κάπου αλλού. Υπενθυμίζουμε ότι μια καθολική μεταβλητή μπορεί να δηλωθεί μία μόνο φορά.

Αρχείο 1 Αρχείο 2

int x,y; extern int x,y;

... ...

static

Η δήλωση static έχει διαφορετικό αποτέλεσμα πάνω σε τοπικές μεταβλητές και διαφορετικό σε καθολικές μεταβλητές.Μία static τοπική μεταβλητή έχει εμβέλεια μέσα στη συνάρτηση που είναι δηλωμένη και κρατά την τιμή της μεταξύ διαδοχικών καλεσμάτων της συνάρτησης. Έτσι μπορεί να χρησιμοποιηθεί σε μία συνάρτηση παραγωγής μιας σειράς αριθμών π.χ.

```
increment(void)

{

static int count=0;

count=count+5;

return (count);

}
```

Η αρχικοποίηση της count γίνεται μία μόνο φορά, στο πρώτο κάλεσμα της increment. Μία καθολική μεταβλητή static έχει εμβέλεια μόνο στο αρχείο στο οποίο είναι δηλωμένη. Αυτό σημαίνει ότι αν και είναι καθολική δεν μπορούν να την δουν ρουτίνες από άλλα αρχεία και ν' αλλάξουν το περιεχόμενό της.Οι μεταβλητές static μας δίνουν την δυνατότητα να αποκρύψουμε ένα μέρος από ένα πρόγραμμα. Αυτό μας βοηθάει πολύ στο γράψιμο μεγάλων προγραμμάτων καθώς επίσης στο γράψιμο συναρτήσεων που θα μπουν σε βιβλιοθήκες.

register

Μία μεταβλητή register αποθηκεύεται σ' ένα καταχωρητή της CPU αντί για την μνήμη όπως με τις απλές μεταβλητές (συνεπώς οι μεταβλητές αυτές δεν έχουν διεύθυνση). Αυτό σημαίνει ότι πράξεις με μεταβλητές τύπου register είναι πολύ πιο γρήγορες εφόσον δεν απαιτείται προσπέλαση στη μνήμη για να δούμε ή ν' αλλάξουμε την τιμή τους. π.χ.

```
register int temp;
```

Δηλώσεις σταθερών

Σταθερές χαρακτήρες γράφονται σε απλά εισαγωγικά. char question_mark = '?'; Σταθερές τύπου string γράφονται μέσα σε διπλά εισαγωγικά, "---". Έτσι συνήθως τυπώνουμε διάφορα μηνύματα. Παραδείγματα σταθερών για τους άλλους τύπους δίνονται στον παρακάτω πίνακα.

int 1234

long int 38754L

short int 123

unsigned int 62222

float 12.345F

float 1.1e-3F

double -0.9876544

char '?'

Υπάρχουν όμως ορισμένοι χαρακτήρες που δεν τυπώνονται με μια σταθερά τύπου string για παράδειγμα τα διπλά εισαγωγικά. Για τους χαρακτήρες αυτούς έχουμε τις λεγόμενες σταθερές backslash που δίνονται στον επόμενο πίνακα.

Κώδικας	**Σημασία**

\b	backspace
\f	form feed
\n	νέα γραμμή
\r	carriage return
\t	οριζόντιο tab
\"	διπλά εισαγωγικά
\'	απλά εισαγωγικά
\0	Null
\\	backslash
\v	κάθετο tab
\a	alert
\N	οκταδική σταθερά
\xN	δεκαεξαδική σταθερά

Τελεστές

Η C έχει τέσσερις τύπους τελεστών:**αριθμητικοί, σύγκρισης (συσχεσιακοί), λογικοί και τελεστές χειρισμού bits (bitwise operators).**Παρακάτω δίνουμε ένα πίνακα με όλους τους τελεστές διατεταγμένους σύμφωνα με την προτεραιότητά τους.

Προτεραιότητα	**Τελεστές**
Υψηλότερη Χαμηλότερη	( ) [ ] -> ! ~ ++ -- -(type) * & sizeof * / % - + << >> < <= > >= == != & ^ | && || ? = += -= *= /=

Συμβατότητα με εκχωρήσεις

Ο τελεστής εκχώρησης έχει την γενική μορφή

<αναγνωριστικό> = <παράσταση>

Κατά την εκχώρηση η τιμή της παράστασης μετατρέπεται στον τύπο της μεταβλητής στο αριστερό μέρος της παράστασης. Για παράδειγμα:

```
int i;

char ch;

float f;

void function(void)

{

ch=i; /* ch= ο χαρακτήρας που αντιστοιχεί στο δεύτερο

byte του i */

i=f; /* i= το ακέραιο μέρος του f */

f=ch; /* f= ο αριθμός που αντιστοιχεί στο ένα byte του ch */

f=i; /* f= ο αριθμός που αντιστοιχεί στα δύο bytes του int */

}
```

Οταν σε μία παράσταση υπάρχουν τελεσταίοι διαφορετικών τύπων τότε μετασχηματίζονται στον τύπο του "ισχυρότερου" τελεσταίου. Στο παρακάτω σχήμα φαίνονται όλοι οι μετασχηματισμοί τύπων που γίνονται κατά τον υπολογισμό της παράστασης

```
r=(ch / i) + (f * d) - (f + i)

------- ------- ------

i d f

-----------

d

----------------

d
```

Μετασχηματισμός του τύπου μιας παράστασης μπορεί να γίνει προσδιορίζοντας τον νέο τύπο μέσα σε παρένθεση πριν από την παράσταση. π.χ.

(τύπος)<παράσταση>

Για παράδειγμα

```
int i;
```

(float) i/2;

Τέλος στη C μπορούμε να έχουμε πολλαπλή εκχώρηση x=y=z=0;

Αριθμητικοί τελεστές

-	αφαίρεση, πρόσημο
+	πρόσθεση
*	πολλαπλασιασμός
/	διαίρεση
%	(mod)
--	ελάττωση μεταβλητής κατά 1
++	αύξηση μεταβλητής κατά 1

Οι τελεστές -- και ++ μπορεί να τοποθετηθούν μπροστά ή μετά ένα τελεσταίο. Η εντολή --x; ισοδυναμεί με την x:=x-1 αλλά η αφαίρεση εκτελείται πριν χρησιμοποιήσουμε την τιμή της x. Όμοια και η εντολή ++x;

Η εντολή x--; ισοδυναμεί με την x:=x-1 αλλά η αφαίρεση εκτελείται αφού χρησιμοποιήσουμε την τιμή της x.

x=3; x=3;

y=++x; y=x++;

αποτέλεσμα: y=4 (x=4) αποτέλεσμα: y=3 (x=4)

Συσχεσιακοί και λογικοί τελεστές

>	Μεγαλύτερο
>=	μεγαλύτερο και ίσον
<	μικρότερο
<=	μικρότερο και ίσον
==	ίσον
!=	διάφορο του ίσον
&&	AND
||	OR
!	NOT

Στη C true είναι μια τιμή διαφορετική του μηδενός και false είναι το μηδέν.

Τελεστές χειρισμού bits

Χειρισμός των bits σημαίνει την δυνατότητα επέμβασης στα bits ενός byte ή μιας λέξης που αντιστοιχούν στους τύπους char και int.

&	AND
|	OR
^	XOR
~	συμπλήρωμα ως προς 1
>>	shift right
<<	shift left

Για παράδειγμα η παράσταση (ch & 127) όπου ch είναι τύπου char εκχωρεί την τιμή 0 στο parity bit.

ch: 1 0 0 1 1 1 0 1

127: 0 1 1 1 1 1 1 1

_____

& 0 0 0 1 1 1 0 1

Ο τελεστής << έχει τον τύπο: variable << number και μετακινεί τα όλα τα bits της μεταβλητής προς τ' αριστερά number θέσεις. Οι κενές θέσεις που δημιουργούνται από τα δεξιά αντικαθίστανται με 0. π.χ.

x=5; x: 0 0 0 0 0 1 0 1

x=x << 2; 0 0 0 1 0 1 0 0

(αποτέλεσμα x=20)

Οι τελεστές πρόσθεσης +, αφαίρεσης -, πολλαπλασιασμού * και διαίρεσης /. Στη διαίρεση πρέπει να έχουμε υπόψη μας ότι η διαίρεση με αριθμούς κινητής υποδιαστολής δίνει αποτέλεσμα του ίδιου τύπου, ενώ η διαίρεση με ακεραίους δίνει μια ακέραια απάντηση.Έτσι, αν η διαίρεση ακεραίων δεν είναι τέλεια, η C απορρίπτει το δεκαδικό μέρος του πηλίκου χωρίς να το στρογγυλοποιεί. Αυτή η διαδικασία λέγεται *αποκοπή*.Όταν ανακατεύουμε ακεραίους με αριθμούς κινητής υποδιαστολής, το αποτέλεσμα είναι αριθμός κινητής υποδιαστολής.

Ακολουθεί ένα ερμηνευτικό παράδειγμα :

```
#include <stdio.h>

main()

{

 printf("ακέραια διαίρεση : 5/3 είναι %d \n", 5/3);

 printf("ακέραια διαίρεση : 8/4 είναι %d \n", 8/4);

 printf("ακέραια διαίρεση : 7/5 είναι %d \n", 7/5);

 printf("διαίρεση κινητής υποδιαστολής : 7./4. είναι %1.2f \n", 7./4.);

 printf("μικτή διαίρεση : 7./4 είναι %1.2f \n", 7./4);
```

}

Το αποτέλεσμα θα είναι :

ακέραια διαίρεση :5/3 είναι 1

ακέραια διαίρεση : 8/4 είναι 2

ακέραια διαίρεση : 7/5 είναι 1

διαίρεση κινητής υποδιαστολής : 7./4. είναι 1.75

μικτή διαίρεση : 7./4 είναι 1.75

Βλέπουμε ότι όσον αφορά τις τρεις πρώτες διαιρέσεις που γίνονται μεταξύ ακεραίων αριθμών, σαν αποτέλεσμα παίρνουμε το *πηλίκο* της ακέραιας διαίρεσης, κάτι που μας είναι χρήσιμο σε πολλές εφαρμογές, ο αντίστοιχος τελεστής δηλ. Του *div* της Pascal. Βλέπουμε ακόμη ότι από το ανακάτεμα ακεραίων και αριθμών κινητής υποδιαστολής, παίρνουμε αριθμό κινητής υποδιαστολής.

Οι Προτεραιότητες των Τελεστών

Η C τοποθετεί κάθε τελεστή σε κάποιο επίπεδο προτεραιότητας. Ο πολλαπλασιασμός και η διαίρεση ανήκουν σε υψηλότερο επίπεδο προτεραιότητας από την πρόσθεση και την αφαίρεση, γι' αυτό και εκτελούνται πρώτα.Αν, όμως, τελεστές του ίδιου επιπέδου προτεραιότητας επιδρούν στον ίδιο τελεστέο, τότε εκτελούνται με τη σειρά εμφάνισής τους στην πρόταση. Για τους περισσότερους τελεστές, η σειρά εκτέλεσης είναι από αριστερά προς τα δεξιά και ο τελεστής = αποτελεί εξαίρεση.

Δηλαδή, στην πρόταση :

*a= 25.0 + 60.0 *b/ c;*

γίνεται πρώτα ο πολλαπλασιασμός 60.0 *b, μετά η διαίρεση αυτού με τη μεταβλητή c και τέλος προστίθεται το 25.0.

Αν, όμως, θελήσουμε να γίνει πρώτα η πρόσθεση και μετά η διαίρεση, τότε πρέπει να βάλουμε παρενθέσεις, ως εξής :

*a= (25.0 + 60.0 *b) / c;*

Ο Τελεστής sizeof

Ο τελεστής αυτός επιστρέφει το μέγεθος σε bytes του τελεστέου του. Ο τελεστέος μπορεί να είναι το όνομα μιας μεταβλητής ή μπορεί να είναι ένας τύπος. Αν είναι τύπος ονόματος, τότε πρέπει ο τελεστέος να μπει μέσα σε παρενθέσεις, αλλιώς οι παρενθέσεις είναι προαιρετικές.

Ακολουθεί ένα παράδειγμα :

```
#include <stdio.h>

main()

{

 int n=10;

 printf("O n έχει %d bytes, όλοι οι ακέραιοι έχουν %d bytes.\n",

 sizeof n, sizeof(int));

}
```

Το αποτέλεσμα θα είναι :

O 10 έχει 2 bytes, όλοι οι ακέραιοι έχουν 2 bytes.

Ο Τελεστής Ακεραίου Υπολοίπου %

Ο τελεστής ακεραίου υπολοίπου (%) χρησιμοποιείται στην αριθμητική των ακεραίων και επιστρέφει το υπόλοιπο της ακέραιας διαίρεσης του ακεραίου στα αριστερά με τον ακέραιο στα δεξιά. Για παράδειγμα, η πράξη 13 % 5 δίνει σαν αποτέλεσμα την τιμή 3, αφού το 5 χωράει δύο φορές στο 13 και έχει υπόλοιπο 3.

Ακολουθεί ένα παράδειγμα :

```
#include <stdio.h>

#define SEC_PER_MIN 60 /* 60 δευτερόλεπτα σ'έναλεπτό*/

main()

{

 int sec, min, left;

 printf("Μετατροπή δευτερολέπτων σε λεπτά και δευτερόλεπτα \n");

 printf("Δώστε τον αριθμό των δευτερολέπτων : \n");

 scanf("%d", &sec);/* διάβασμα του αριθμού των δευτερολέπτων */

min = sec / SEC_PER_MIN; /*αριθμόςλεπτών*/

left = sec % SEC_PER_MIN; /*αριθμός δευτερολέπτων που έμειναν*/

printf(" %d δευτερόλεπτα είναι %d λεπτά και %d δευτερόλεπτα.\n",

sec, min, left);
```

}

Το αποτέλεσμα θα είναι :

Μετατροπή δευτερολέπτων σε λεπτά και δευτερόλεπτα

Δώστε τον αριθμό των δευτερολέπτων :

152

152 δευτερόλεπτα είναι 2 λεπτά και 32 δευτερόλεπτα.

Όπως είδαμε στο παραπάνω παράδειγμα, για να υπολογίσουμε πόσα λεπτά υπάρχουν σ' έναν αριθμό δευτερολέπτων sec, παίρνουμε το πηλίκο της ακέραιας διαίρεσης του sec με το SEC_PER_MIN, που είναι ουσιαστικά το 60 και για να βρούμε πόσα δευτερόλεπτα περίσσευσαν που δεν χώρεσαν σ' ένα λεπτό, παίρνουμε το υπόλοιπο της ακέραιας διαίρεσης του sec με το SEC_ PER_MIN.

Ο Τελεστής Αύξησης ++

Ο τελεστής αύξησης (++) αυξάνει κατά ένα την τιμή του τελεστέου. Ο τελεστής αυτός υπάρχει σε δύο μορφές : Το ++ μπορεί να βρίσκεται πριν από την επηρεαζόμενη μεταβλητή και ονομάζεται τελεστής *προγενέστερης αύξησης* ή μετά απ' αυτήν, οπότε ονομάζεται τελεστής *μεταγενέστερης αύξησης*.

Τα δύο είδη αυξήσεων διαφέρουν στον χρόνο που γίνεται η αύξηση. Πρώτα, όμως, θα δούμε τις ομοιότητές τους και αργότερα τις διαφορές.

Ακολουθεί ένα παράδειγμα :

#include <stdio.h>

main()

{

int a = 0, b = 0;

while (a < 5)

{

a++;

++b;

printf("b = %d, a = %d \n", b, a);

}

```
}
```

Το αποτέλεσμα θα είναι :

```
b= 1,a= 1

b= 2,a= 2

b= 3,a= 3

b= 4,a= 4

b= 5,a= 5
```

Θα μπορούσαμε να πάρουμε το ίδιο αποτέλεσμα χρησιμοποιώντας τις εξής εντολές :

```
b=a+ 1;

a=a+ 1;
```

Ο λόγος που χρησιμοποιούμε αυτόν τον τελεστή αύξησης είναι ότι έχει κάποια πλεονεκτήματα που θα φανούν αργότερα. Και για να γίνουμε πιο σαφείς, ακολουθεί ένα παράδειγμα :

```
euro = 2.0;

while (++euro < 100.00)

{

draxmes = 340.75 * euro;

printf("%10.2f %20.2f\n", euro, draxmes);

}
```

Εδώ έχουμε συνδυάσει τη διαδικασία αύξησης και σύγκρισης του βρόχου while σε μία μόνο έκφραση, δηλ. η τιμή της μεταβλητής euro πρώτα αυξάνεται κατά 1 και αμέσως μετά συγκρίνεται με το 100.00. Έτσι, έχουμε τον έλεγχο του βρόχου και την αύξηση της μεταβλητής στο ίδιο μέρος.

Ας δούμε ακόμα ένα παράδειγμα :

```
#include <stdio.h>

main()

{

int a =1, b = 1;

int aplus, plusb;
```

*aplus = a++; /*μεταγενέστερος*/*

*plusb = ++b; /*προγενέστερος*/*

printf("a aplus b plusb \n");

printf("%1d %5d %5d %5d\n", a, aplus, b, plusb);

}

Το αποτέλεσμα θα είναι :

a aplus b plusb

2 1 2 2

Τόσο η a όσο και η b αυξάνονται κατά ένα. Επομένως, η aplus έχει την τιμή της a *πριν* η a αλλάξει και η plusb έχει την τιμή της b *μετά* την αλλαγή της b.

aplus = a++;

/ μεταγενέστερη : η a άλλαξε μετά τη χρήση της τιμής της */*

plusb = ++b;

/ προγενέστερη : η b άλλαξε πριν τη χρήση της τιμής της */*

Αν ένας από τους τελεστές αύξησης χρησιμοποιείται μόνος του σε κάποια πρόταση, τότε δεν έχει σημασία ποια μορφή θα χρησιμοποιήσουμε. Η επιλογή, όμως, έχει σημασία όταν ο τελεστής και ο τελεστέος αποτελούν μέρη κάποιας έκφρασης, όπως μόλις είδαμε.

Για να συνοψίσουμε, το n++ σημαίνει :

"κάνε χρήση του n και μετά αύξησέ το"

και το ++n σημαίνει :

"αύξησε το n και μετά χρησιμοποίησέ το".

Ο Τελεστής Μείωσης - -

Δεν θα ήταν δυνατόν φυσικά να μην υπήρχε και ο αντίστοιχος τελεστής μείωσης, δηλ. αντί για το ++ χρησιμοποιούμε το - -.

- -count; / τελεστής προγενέστερης μείωσης */*

count- -; / τελεστής μεταγενέστερης μείωσης */*

Ακολουθεί κι ένα παράδειγμα με την χρήση του τελεστή μείωσης :

#include <stdio.h>

```
#define MAX 100

main()

{

int count = MAX + 1;

 while (- -count > 0)

{

printf("Υπάρχουν %d βιβλία στη βιβλιοθήκη \n", count);

printf("Πάρε ένα βιβλίο, \n");

printf("%d βιβλία έμειναν στη βιβλιοθήκη \n\n", count-1);

}

}
```

Το αποτέλεσμα θα είναι :

Υπάρχουν 100 βιβλία στη βιβλιοθήκη

Πάρε ένα βιβλίο,

99 βιβλία έμειναν στη βιβλιοθήκη

Υπάρχουν 99 βιβλία στη βιβλιοθήκη

Πάρε ένα βιβλίο,

98 βιβλία έμειναν στη βιβλιοθήκη

Οι τελεστές αύξησης και μείωσης έχουν υψηλή προτεραιότητα και μόνο οι παρενθέσεις έχουν υψηλότερη. Πρέπει πάντως να αποκτήσει κανείς αρκετή εμπειρία με τους τελεστές αύξησης και μείωσης, να μάθει τις ιδιαιτερότητες της έκδοσης της C με την οποία δουλεύει και μετά να τους χρησιμοποιεί.

Πρέπει να έχουμε υπόψη μας τα εξής :

- Δεν πρέπει να χρησιμοποιούμε τελεστές αύξησης ή μείωσης σε μεταβλητή που αποτελεί μέρος περισσότερων του ενός ορισμάτων ή συναρτήσεων.

- Δεν πρέπει να χρησιμοποιούμε τελεστές αύξησης ή μείωσης σε μεταβλητή που εμφανίζεται περισσότερες από μία φορές σε μια έκφραση.

Δηλαδή, οι παρακάτω εκφράσεις μπορεί να μπερδέψουν τη C και πάντως κανείς δεν εγγυάται για το αποτέλεσμά τους :

*printf("%10d %10d\n", num, num*num++);*

a=num/2 + 5(1 +num++);*

b=n++ +n++;

Εκφράσεις και Προτάσεις

Μια *έκφραση* αποτελείται από έναν συνδυασμό τελεστών και τελεστέων (ο τελεστέος είναι αυτό, πάνω στο οποίο δρα ο τελεστής). Η απλούστερη έκφραση είναι ένας μόνο τελεστέος.

Ακολουθούν μερικές εκφράσεις :

4

4 + 21

a(b + c/d)/20*

*q = 5*2*

x = ++q % 3

q > 3

Οι τελεστέοι μπορεί να είναι σταθερές, μεταβλητές ή συνδυασμοί αυτών των δύο. Κάθε έκφραση στη C έχει μια τιμή και για να βρούμε την τιμή αυτή, εκτελούμε τις πράξεις με τη σειρά που υποδεικνύεται από την προτεραιότητα των τελεστών. Εκφράσεις σχέσεων, όπως η q > 3, έχουν τιμή 1 αν είναι αληθείς και 0 αν είναι ψευδείς.

Ακολουθούν μερικές εκφράσεις :

-4+6= 2

c = 3 + 8= 11

5>3 1

6+(c=3+8) 17

Οι *προτάσεις* είναι τα πρωταρχικά δομικά στοιχεία ενός προγράμματος, δηλ. το πρόγραμμα είναι μια σειρά προτάσεων και σημείων στίξης. Στη C οι προτάσεις κλείνουν με το σύμβολο ;. Έτσι, η a = 4 είναι μια έκφραση, αλλά η a= 4; είναι μια πρόταση. Στο παράδειγμα : x = 6 + (y=5); η υποέκφραση y=5 είναι μια

ολοκληρωμένη εντολή, αλλά απλά είναι μέρος μιας πρότασης. Επειδή συνεπώς μια ολοκληρωμένη εντολή δεν είναι και απαραίτητα μια πρόταση, το σύμβολο (;) χρειάζεται για να χαρακτηρίζει τις εντολές που είναι αληθινές προτάσεις.

Στο επόμενο παράδειγμα χρησιμοποιούμε τέσσερα είδη προτάσεων:

```
#include <stdio.h>

main() /* βρίσκει το άθροισμα των 20 πρώτων ακεραίων */

{

int count, sum; /* πρόταση δήλωσης */

count = 0; /* πρόταση καταχώρησης */

sum = 0;/* πρόταση καταχώρησης */

while (count++ < 20)/*πρόταση ελέγχου while */

sum = sum + count; /*πρόταση*/

printf("άθροισμα = %d\n", sum); /* πρόταση - συνάρτηση */

}
```

Μια *σύνθετη πρόταση* αποτελείται από δύο ή περισσότερες προτάσεις που περικλείονται από αγκύλες. Λέγεται επίσης και *μπλοκ (block)*.

Ακολουθούν παραδείγματα :

```
/* τμήμα προγράμματος 1 */

index = 0;

while (index++ < 10)

sum = 10*index + 2;

printf("sum = %d\n", sum);

 /* τμήμα προγράμματος 2 */

index = 0;

while (index++ < 10)

{

sum = 10*index + 2;

printf("sum = %d\n", sum);
```

}

Στο τμήμα προγράμματος 1, ο βρόχος while περιλαμβάνει μόνο μια πρόταση αντικατάστασης, δηλ. όταν λείπουν τα { και }, μια πρόταση while τρέχει από το while μέχρι το επόμενο σύμβολο ;.

Στο τμήμα προγράμματος 2, τα σύμβολα { και } δηλώνουν ότι και οι δύο προτάσεις αποτελούν μέρος του βρόχου while. Ολόκληρη η σύνθετη πρόταση θεωρείται σαν μια απλή πρόταση με την έννοια της δομής της πρότασης while.

Οι Μετατροπές Τύπου

Η C δεν γκρινιάζει τόσο εύκολα όσο η Pascal όταν ανακατεύουμε μεταβλητές και σταθερές διαφορετικών τύπων δεδομένων. Η C χρησιμοποιεί ορισμένους κανόνες για να μετατρέψει αυτόματα τους τύπους :

1. Όταν εμφανίζονται σε εκφράσεις, τόσο ο τύπος *char* όσο και ο τύπος *short*, με πρόσημο ή χωρίς, αυτόματα μετατρέπονται σε τύπο *int*. Επειδή αυτές είναι μετατροπές προς κάποιο μεγαλύτερο τύπο, λέγονται *προαγωγές*.

2. Σε κάθε πράξη όπου εμπλέκονται δύο τύποι, οι δύο τιμές μετατρέπονται στον τύπο αυτής με τον "υψηλότερο" βαθμό.

3. Η ιεραρχία των τύπων από τους υψηλότερους προς τους χαμηλότερους είναι η εξής : long double, double, float, unsigned long, long, unsigned int και int.

4. Σε μια πρόταση καταχώρησης, το τελικό αποτέλεσμα των υπολογισμών μετατρέπεται στον τύπο της μεταβλητής στην οποία καταχωρήθηκε η τιμή. Έτσι, όμως, μπορεί μια τιμή να μετατραπεί σε τύπο χαμηλότερου βαθμού, όταν π.χ. καταχωρούμε τύπο float σε τύπο int και γίνεται, όπως είδαμε, στρογγυλοποίηση του αριθμού.

Ακολουθεί ένα παράδειγμα :

#include <stdio.h>

main()

{

char ch;

int i;

float fl;

fl = i = ch = 'A';

printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);

ch = ch + 1;

*i = fl + 2 * ch;*

*fl = 2.0 * ch + i;*

printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);

}

Το αποτέλεσμα θα είναι :

ch = A, i = 65, fl = 65.00

ch = B, i = 197, fl = 329.00

Ο Τελεστής Εκμαγείο

Όλες οι μετατροπές τύπων που αναφέραμε μέχρι τώρα γίνονται αυτόματα. Όμως, είναι πιθανό να θέλουμε να δώσουμε εμείς τις οδηγίες για την ακριβή μετατροπή του τύπου που θέλουμε. Η μέθοδος αυτή λέγεται *εκμαγείο* και συνίσταται στην τοποθέτηση μπροστά από την ποσότητα του ονόματος, του επιθυμούμενου τύπου μέσα σε παρενθέσεις. Οι παρενθέσεις μαζί με το όνομα του τύπου αποτελούν τον *τελεστή-εκμαγείο*.

Ακολουθούν παραδείγματα, όπου η μεταβλητή m είναι τύπου int :

m = 1.6 + 1.7;

m = (int) 1.6 + (int) 1.7;

Το πρώτο παράδειγμα κάνει αυτόματη μετατροπή και η μεταβλητή m παίρνει την τιμή 3. Το δεύτερο παράδειγμα περιέχει δύο εκμαγεία τύπου int και οι μετατροπές σε ακεραίους γίνονται πριν από την πρόσθεση και έτσι η τιμή της μεταβλητής m είναι 2.

Ορίσματα Συναρτήσεων και Μετατροπές Τύπων

Το επόμενο παράδειγμα περιέχει μια συνάρτηση που τυπώνει έναν ορισμένο αριθμό συμβόλων #. Το παράδειγμα αυτό δείχνει ακόμα μερικά σημεία που αφορούν μετατροπές τύπου.

#include<stdio.h>

main()

{

```
int times =4;

char ch = ‘!’; /* ο ASCIIκώδικας είναι 33 */

float f =5.0;

f(times); /* όρισμα τύπου int */

f(ch); /*char αυτόματα ®int*/

f((int)f);/* το εκμαγείο αναγκάζει τηνf®int */

}

f(n) /* παλιός τρόπος επικεφαλίδας συνάρτησης */

 int n; /* η συνάρτηση έχει ένα όρισμα τύπου int */

{

while(n- - > 0)

printf("#");

printf("\n");

}
```

Το αποτέλεσμα θα είναι :

```
####

##################################

#####
```

Εφόσον η συνάρτηση f() παίρνει ένα όρισμα, βάζουμε ένα όνομα μεταβλητής : n. Μετά δηλώνουμε τον τύπο της n. Αυτές οι δηλώσεις ορισμάτων βρίσκονται ανάμεσα στο όνομα της συνάρτησης και στην αρχική αγκύλη. Η δήλωση ενός ορίσματος δημιουργεί μια μεταβλητή που λέγεται *τυπικό όρισμα* ή *τυπική παράμετρος*.

Στο πρόγραμμά μας, η κλήση f(times) καταχωρεί την τιμή της μεταβλητής times, δηλ. το 4 στην n. Λέμε ότι η κλήση μιας συνάρτησης περνά (μεταβιβάζει) μια τιμή και αυτή η τιμή λέγεται *ενεργό όρισμα* ή *ενεργή παράμετρος*.

Τα ονόματα μεταβλητών ανήκουν αποκλειστικά στην συνάρτηση, που σημαίνει ότι ένα όνομα που ορίστηκε σε μια συνάρτηση δεν έχει σχέση με το ίδιο όνομα που ορίστηκε κάπου αλλού. Αυτό σημαίνει ότι μπορεί να έχουμε δύο μεταβλητές με το ίδιο όνομα και το πρόγραμμα να εξακολουθεί να διακρίνει ποια είναι ποια.

Η τελευταία κλήση f((int) f) χρησιμοποιεί ένα εκμαγείο τύπου για να μετατρέψει το f στον κατάλληλο τύπο γι' αυτό το όρισμα. Αν δεν χρησιμοποιούσαμε αυτήν την μετατροπή, τότε η συνάρτηση θα περίμενε να βρει τιμή τύπου int και θα διάβαζε έτσι μόνο τα 2 bytes από τα 8 bytes της τιμής τύπου float. Έτσι, προσαρμόσαμε τον τύπο της μεταβλητής με τον τύπο του ορίσματος της συνάρτησης.

Η Εντολή While

Η γενική μορφή (σύνταξη) της εντολής while είναι η εξής :

while*(έκφραση)*

πρόταση

Η έκφραση είναι μια σύγκριση τιμών και έχει γενικά ένα λογικό αποτέλεσμα. Η πρόταση μπορεί να είναι μια απλή πρόταση με το σύμβολο ; στο τέλος της ή μια σύνθετη πρόταση κλεισμένη ανάμεσα στις αγκύλες { και }. Αν η έκφραση είναι αληθής (δηλ. γενικότερα μη μηδενική), η πρόταση εκτελείται μία φορά και συνεχίζει μέχρι η έκφραση να γίνει ψευδής (δηλ. γενικότερα μηδενική).Φυσικά, θα πρέπει η τιμή της έκφρασης ελέγχου να αλλάζει τιμή, έτσι ώστε η έκφραση τελικά να γίνει ψευδής και να βγούμε από τον βρόχο. Υπάρχει, βέβαια, και η περίπτωση η έκφραση να είναι ψευδής και να μην μπούμε καθόλου μέσα στο σώμα του βρόχου.

Ακολουθεί ένα παράδειγμα με μια εντολή while:

while (scanf("%d", &num) == 1)

; / παραλείπει την είσοδο που είναι ακέραιος */*

το τμήμα αυτό του προγράμματος παραλείπει (αγνοεί) τις ακέραιες τιμές και συνεχίζει μόνο όταν συναντήσει μια μη ακέραια τιμή.

Ένα Παράδειγμα με την While

Θα δούμε τώρα τον βρόχο while σ' ένα πρόγραμμα που αθροίζει ακέραιες τιμές που εισάγονται από το πληκτρολόγιο.

#include <stdio.h>

main()

{

long num;

long sum= 0L; / αρχική τιμή της sum= 0 */*

```
int status;
printf("Δώστε έναν ακέραιο. ");
printf("Δώστε q για να σταματήσετε. \n");
status=scanf(" %ld", &num);
while(status== 1) /* το == σημαίνει 'ίσο με' */
{
sum=sum+num;
printf("Δώστε τον επόμενο ακέραιο. ");
printf("Δώστε q για να σταματήσετε. \n");
status=scanf(" %ld", &num);
}
printf("Οι ακέραιοι αριθμοί έχουν άθροισμα %ld. \n",sum);
}
```

Το αποτέλεσμα θα είναι :

Δώστε έναν ακέραιο. Δώστε q για να σταματήσετε. 20

Δώστε τον επόμενον ακέραιο. Δώστε q για να σταματήσετε. 5

Δώστε τον επόμενον ακέραιο. Δώστε q για να σταματήσετε. 30

Δώστε τον επόμενον ακέραιο. Δώστε q για να σταματήσετε. Q

Οι ακέραιοι αριθμοί έχουν άθροισμα 55.

Ας δούμε λίγες ενδιαφέρουσες λεπτομέρειες του προγράμματος. Κατ' αρχήν, ο τελεστής == είναι ο *τελεστής ισότητας* της C, ενώ η εντολή status=1 θα καταχωρούσε το 1 στη μεταβλητή status. Το πρόγραμμα τελειώνει όταν η status πάρει μια τιμή διαφορετική από το 1, οπότε σταματάει ο βρόχος και το πρόγραμμα εμφανίζει την τελική τιμή του αθροίσματος. Βλέπουμε, όμως, ότι με την χρήση της scanf() πετύχαμε δύο πράγματα : διαβάζουμε μια καινούργια τιμή για την μεταβλητή num και ταυτόχρονα κάνουμε χρήση της επιστρεφόμενης τιμής της scanf(). Η scanf() επιστρέφει τον αριθμό των στοιχείων που διάβασε με επιτυχία. Αν, δηλ., διαβάσει έναν ακέραιο, τότε επιστρέφει την τιμή 1, η οποία και καταχωρείται στην μεταβλητή status, αλλιώς αν δώσουμε μια μη αριθμητική είσοδο, όπως το q, τότε κανένα στοιχείο δεν διαβάζεται και η επιστρεφόμενη τιμή της status γίνεται ίση με 0. Έτσι,

με την χρήση της scanf() πετύχαμε μαζί με το διάβασμα μιας τιμής και τον έλεγχο της εξόδου από τον βρόχο. Βλέπουμε ακόμη ότι έχουμε μια εντολή scanf() πριν μπούμε στον βρόχο while, για να μπορέσει έτσι να γίνει ο αρχικός έλεγχος της συνθήκης και επίσης μια εντολή scanf() στο τέλος του βρόχου για να μπορούν να συνεχίζονται οι επαναλήψεις.

Θα μπορούσαμε να γράψουμε την εντολή while και ως εξής :

while (scanf("%ld", &num)==1)

Οι Αληθείς και οι Ψευδείς Τιμές

Ακολουθεί ένα παράδειγμα :

```
#include <stdio.h>

main()

{

int true, false;

true = (5>2); /* τιμή μιας αληθούς σχέσης */

false = (5==2);  /* τιμή μιας ψευδούς σχέσης */

printf("αληθής = %d; ψευδής = %d \n", true, false);

}
```

Το αποτέλεσμα του προγράμματος θα είναι :

αληθής = 1; ψευδής = 0

Δηλαδή, στη C μια αληθής έκφραση έχει τιμή 1 και μια ψευδής έκφραση έχει τιμή 0.

Ο παρακάτω βρόχος δεν σταματάει ποτέ :

```
while (1)

{

 ...

}
```

Από το επόμενο παράδειγμα συμπεραίνουμε ότι στη C αληθείς θεωρούνται όλες οι μη μηδενικές τιμές και μόνο το 0 αναγνωρίζεται σαν ψευδές. Έτσι, ουσιαστικά στη C γίνεται αριθμητικός έλεγχος και όχι έλεγχος αληθούς/ψευδούς.

```
#include <stdio.h>
```

```
main()
{
int n = 3;
while (n)
printf("%d\n", n- -);
n = -3;
while (n)
printf("%2d\n", n++);
}
```

Το αποτέλεσμα θα είναι :

```
3
2
1
-3
-2
-1
```

Έτσι, στη C η έκφραση *while (a!=0)* θα μπορεί να γραφεί και ως εξής :

```
while (a)
```

Ο Βρόχος For

Ο βρόχος for αποφεύγει τα τρία βήματα ενός βρόχου while, δηλ. την απόδοση αρχική τιμής, τον έλεγχο της συνθήκης τερματισμού και την αλλαγή της τιμής, όταν, όπως ξέρουμε από τις άλλες γλώσσες προγραμματισμού, οι επαναλήψεις που πρέπει να κάνουμε είναι ορισμένες σε αριθμό. Στη C, όμως, αυτό δεν είναι υποχρεωτικό. Η εντολή for χρησιμοποιεί τρεις εκφράσεις ελέγχου, χωρισμένες με αγγλικές άνω-τελείες (;), για να ελέγχουν τη διαδικασία της ανακύκλωσης. Η πρόταση αρχικών τιμών εκτελείται μόνο μία φορά πριν κάποια από τις προτάσεις του βρόχου εκτελεστεί. Αν η έκφραση ελέγχου είναι αληθής (ή μη-μηδενική), ο βρόχος εκτελείται μία φορά. Μετά, υπολογίζεται η έκφραση ανανέωσης και η έκφραση ελέγχου ελέγχεται μια φορά ακόμα. Η πρόταση *for* είναι ένας βρόχος συνθήκης εισόδου, που σημαίνει, όπως αναφέρθηκε προηγουμένως, ότι η απόφαση για μια

ακόμα εκτέλεση του βρόχου, παίρνεται πριν την εκτέλεση αυτή. Έτσι, είναι πιθανό ο βρόχος να μην εκτελεστεί ποτέ. Το μέρος των προτάσεων μπορεί να αποτελείται από μια απλή πρόταση ή από μια σύνθετη πρόταση.

Η σύνταξη της εντολής for είναι ως εξής :

for *(αρχική τιμή; Έλεγχος; Ανανέωση)*

πρόταση

Ο βρόχος επαναλαμβάνεται μέχρι ο *έλεγχος* να γίνει ψευδής ή μηδέν.

Ακολουθεί ένα παράδειγμα :

```
#include <stdio.h>

#define NUMBER 10

main()

{

int count;

for (count=1; count<=NUMBER; count++)

printf("Florina per sempre!\n");

}
```

Οι παρενθέσεις που ακολουθούν τη λέξη-κλειδί for περιέχουν τρεις εκφράσεις χωρισμένες με το σύμβολο ;. Η πρώτη έκφραση δίνει αρχική τιμή και εκτελείται μία φορά μόνο στην αρχή. Η δεύτερη έκφραση είναι μια συνθήκη ελέγχου και υπολογίζεται πριν από κάθε δυναμική εκτέλεση του βρόχου. Ο βρόχος τελειώνει όταν η έκφραση γίνει ψευδής. Η τρίτη έκφραση υπολογίζεται στο τέλος κάθε βρόχου. Η πρόταση for ακολουθείται από μια απλή ή σύνθετη πρόταση.

Μπορούμε να καθυστερήσουμε για λίγο το τρέξιμο ενός προγράμματος, ως εξής :

```
for (n=1; N<=10000; N++)

 ;
```

Η ANSI C διαθέτει και τη συνάρτηση *clock()*, που μπορεί να χρησιμοποιηθεί για να δημιουργήσουμε χρονικές καθυστέρησης.

Ευέλικτες Χρήσεις της Εντολής For

- Ο ατέρμονας βρόχος με την for γίνεται ως εξής :*for (; ;).*
- Η εντολή for της C μπορεί να χρησιμοποιήσει και αύξηση διάφορη του 1 :

for (n=0; N<60; N=n+10)

- Ακόμα, μπορούμε να μετράμε με χαρακτήρες αντί για αριθμούς :

for (ch='a'; ch<='z'; ch++)

- Μπορούμε ακόμα να ελέγχουμε κάποιες άλλες συνθήκες, εκτός από τον γνωστό μας αριθμό των επαναλήψεων :

*for (num=1; num*num*num<=216; num++)*

- Μπορούμε ακόμα να κάνουμε πράξεις με την ποσότητα ελέγχου :

*for (d=100.0; d<150.0; d=d*1.1)*

- Μπορούμε γενικά να χρησιμοποιήσουμε οποιαδήποτε νόμιμη έκφραση για την τρίτη έκφραση :

*for (x=1; Y<=75; Y=++x*5+50)*

- Η πρώτη έκφραση δεν χρειάζεται να δίνει πάντα αρχική τιμή σε μια μεταβλητή :

for (printf(…); num!=6; scanf("%d", &num))

- Μπορούμε να έχουμε περισσότερους προσδιορισμούς αρχικής τιμής ή ανανεώσεις, χωρισμένες με κόμμα, μέσα στον βρόχο for :

for (x=1, y=2; X<=16; X++, y=y+3)

Συμπεραίνουμε ότι πρόταση for δεν χρησιμοποιείται μόνο για ένα αυστηρά συγκεκριμένο πλήθος επαναλήψεων, αλλά μπορεί να κάνει πολλά περισσότερα πράγματα.

Περισσότεροι Τελεστές Καταχώρησης

Έχουμε ήδη διαπιστώσει ότι η C έχει αρκετούς τελεστές κατχώρησης. Ο πιο βασικός είναι ο =, ο οποίος απλά καταχωρεί την τιμή της έκφρασης που βρίσκεται στα δεξιά του στην μεταβλητή που βρίσκεται στα αριστερά του. Οι άλλοι τελεστές καταχώρησης ανανεώνουν μεταβλητές : +=, –=, *=, /=, %=.

Κάθε ένας από αυτούς τους τελεστές χρησιμοποιείται μ' ένα όνομα μεταβλητής στα αριστερά του και μια έκφραση στα δεξιά του. Στην μεταβλητή καταχωρείται μια νέα τιμή ανάλογα με την τιμή της έκφρασης στα δεξιά. Η ακριβής ρύθμιση εξαρτάται από τον τελεστή. Για παράδειγμα :

a+= 20 είναι το ίδιο όπως a=a+ 20

b–= 2 είναι το ίδιο όπως b =b– 2

c= 2 είναι το ίδιο όπως c=c* 2*

d/= 2.5 είναι το ίδιο όπως d=d/ 2.5

e%= 3 είναι το ίδιο όπως e=e% 3

x= 3 *y+12 είναι το ίδιο όπως x=x* (3 *y+ 12)*

Ο Βρόχος Do While

Είδαμε ότι ο βρόχος while και ο βρόχος for είναι και οι δύο βρόχοι συνθήκης εισόδου. Η C έχει και έναν *βρόχο συνθήκης εξόδου*, στον οποίο η συνθήκη ελέγχεται μετά από κάθε επανάληψη του βρόχου. Αυτός είναι ο βρόχος *do while*. Η πρόταση do while δημιουργεί έναν βρόχο που επαναλαμβάνεται μέχρι η συνθήκη ελέγχου να γίνει ψευδής ή μηδέν. Ο βρόχος do while είναι ένας *βρόχος συνθήκης εξόδου*, δηλ. η απόφαση για μια ακόμα επανάληψη παίρνεται μετά από κάθε εκτέλεση του βρόχου. Έτσι, ο βρόχος πρέπει να εκτελεστεί τουλάχιστον μία φορά. Το μέρος των προτάσεων μπορεί να αποτελείται είτε από μία απλή πρόταση ή από μία σύνθετη.

```
do
scanf("%d", &number)
while(number!=20);
```

Ακολουθεί ένα παράδειγμα :

```
#include <stdio.h>
main()
{
char ch;
do
{
scanf("%c", &ch);
```

printf("%c", ch);

} while (ch!='#');

}

Βλέπουμε ότι το πρόγραμμα διαβάζει χαρακτήρες και τους τυπώνει μέχρι να εμφανιστεί ο χαρακτήρας #. Θα τυπώσει, όμως, και τον χαρακτήρα #, όταν αυτός εμφανιστεί. Για να αποφύγουμε αυτή την περίπτωση, μπορούμε να χρησιμοποιήσουμε την εντολή if, ως εξής :

if (ch<>'#')

printf("%c",ch);

Η γενική μορφή του βρόχου do while είναι η εξής :

do

πρόταση

while*(έκφραση);*

Το μέρος *πρόταση* επαναλαμβάνεται μέχρι η *έκφραση* να γίνει ψευδής ή μηδέν.

Ένας βρόχος do while εκτελείται πάντα τουλάχιστον μία φορά, αφού ο έλεγχος γίνεται μετά την εκτέλεση του σώματος του βρόχου. Από την άλλη μεριά, ένας βρόχος for ή ένας βρόχος while μπορεί να μην εκτελεστεί καμία φορά, αφού ο έλεγχος γίνεται πριν από την εκτέλεση της εντολής. Το φαινομενικό αυτό ελάττωμα που έχει ο βρόχος do while, ότι δηλ. εκτελείται τουλάχιστον μία φορά και ίσως και όταν δεν χρειάζεται, είδαμε ότι πολύ εύκολα μπορεί να διορθωθεί με χρήση της εντολής if μέσα στον βρόχο.

Οι Εσωτερικοί Βρόχοι

Εσωτερικός βρόχος λέγεται αυτός που βρίσκεται στο εσωτερικό ενός άλλου βρόχου. Μια συνήθης χρήση των εσωτερικών βρόχων είναι για την απεικόνιση δεδομένων σε γραμμές και στήλες. Ο ένας βρόχος χειρίζεται όλες τις στήλες σε μια γραμμή και ο άλλος όλες τις γραμμές.

Ακολουθεί ένα παράδειγμα :

#include <stdio.h>

#define GRAMMES 5

#define STILES 5

```
main()
{
int row;
char ch;
for (row=0; row<GRAMMES; row++)
{
for (ch='A'; ch<'A'+STILES; ch++)
printf("%c", ch);
printf("\n");
}
}
```

Το αποτέλεσμα θα είναι :

```
ABCDE
ABCDE
ABCDE
ABCDE
 ABCDE
```

Ο πρώτος βρόχος for λέγεται *έξω βρόχος* και ο άλλος βρόχος for λέγεται *μέσα βρόχος*. Ο έξω βρόχος αρχίζει με τη row να έχει τιμή 0 και τελειώνει όταν η row γίνει ίση και με το 4. Άρα, κάνει 5 επαναλήψεις. Η πρώτη πρόταση σε κάθε επανάληψη είναι ο άλλος βρόχος for. Αυτός ο βρόχος κάνει επίσης 5 κύκλους, τυπώνοντας τους χαρακτήρες από το Α μέχρι το Ε στην ίδια γραμμή.

Μόλις τελειώσει ο μέσα βρόχος, ακολουθεί μια πρόταση printf("\n"), που όπως ξέρουμε αλλάζει γραμμή. Βλέπουμε συνεπώς, ότι ο μέσα βρόχος τυπώνει 5 χαρακτήρες σε μια γραμμή και ο έξω βρόχος δημιουργεί 5 γραμμές. Μπορεί, όμως, ο μέσα βρόχος να εξαρτάται από τον έξω βρόχο και να συμπεριφέρεται έτσι διαφορετικά σε κάθε επανάληψη.

Ακολουθεί το προηγούμενο παράδειγμα τροποποιημένο :

```
#include <stdio.h>
```

```
#define GRAMMES 5

#define STILES 5

main()

{

int row;

char ch;

for (row=0; row<GRAMMES; row++)

{

for (ch='A'+row; ch<'A'+STILES; ch++)

printf("%c", ch);

printf("\n");

}

}
```

Το αποτέλεσμα θα είναι :

```
ABCDE

BCDE

CDE

DE

E
```

Βλέπουμε, δηλ., ότι αλλάζει σε κάθε εκτέλεση του εσωτερικού βρόχου η αρχική τιμή του βρόχου αυτού, ενώ η συνθήκη ελέγχου και η αύξηση μένουν οι ίδιες.

Οι Πίνακες

Γνωρίζουμε την μεγάλη σημασία που έχουν οι πίνακες για τον προγραμματισμό. Μπορούμε να αποθηκεύουμε αρκετά στοιχεία για παρόμοιες πληροφορίες μ' έναν πολύ βολικό τρόπο.Ένας πίνακας είναι μια σειρά δεδομένων του ίδιου τύπου, όπως π.χ. 15 στοιχεία τύπου char ή 10 στοιχεία τύπου int, αποθηκευμένα ακολουθιακά. Ολόκληρος ο πίνακας έχει ένα απλό όνομα και τα δεδομένα του, που λέγονται *στοιχεία του πίνακα*, μπορούν να προσπελαστούν με χρήση μιας κατάστασης ακεραίων.

Για παράδειγμα, η παρακάτω δήλωση : float pin[20];, δηλώνει ότι ο pin είναι ένας πίνακας με 20 στοιχεία, με πρώτο στοιχείο το pin[0], δεύτερο στοιχείο το pin[1], κοκ μέχρι και το pin[19]. Βλέπουμε ότι η αρίθμηση των στοιχείων του πίνακα αρχίζει από το 0 και όχι από το 1. Σε κάθε στοιχείο του παραπάνω πίνακα μπορούμε να καταχωρήσουμε μια τιμή τύπου float, ως εξής :

```
pin[5] = 20.16;

pin[6] = 1.3e+12;
```

Μπορούμε να ορίσουμε και πίνακες άλλων τύπων :

```
int numbers[10];

char alpha[26];

long bignumbers[100];
```

Οι αριθμοί που χρησιμοποιούνται για να χαρακτηρίζουν τα στοιχεία του πίνακα καλούνται *δείκτες του πίνακα*, οι οποίοι πρέπει να είναι ακέραιοι και να αρχίζουν από το 0. Τα στοιχεία του πίνακα είναι αποθηκευμένα στην μνήμη, το ένα δίπλα στο άλλο. Ακολουθεί ένα πρόγραμμα με πίνακες και με χρήση της εντολής for, το οποίο διαβάζει 10 βαθμούς, τους εμφανίζει στην οθόνη και μετά υπολογίζει το άθροισμά τους και τον μέσο όρο τους :

```
#include <stdio.h>

#define SIZE 10

main()

{

int i, sum, b[SIZE];

float average;

printf("Δώστε %dβαθμούς \n"SIZE);

for (i = 0; i < SIZE; i++)

scanf("%d", &b[i]); /* διάβασμα των 10 βαθμών */

printf("Οι βαθμοί που διαβάστηκαν είναι οι εξής : \n");

for (i = 0; i < SIZE; i++)

printf("%5d",b[i]); /* επιβεβαίωση της καταχώρησης */
```

```
printf("\n");

for (i = 0; i < SIZE; i++)

sum+=b[i]; /* άθροιση των βαθμών */

average= (float)sum/SIZE; /* υπολογισμός μέσους όρου */

printf("Άθροισμα βαθμών = %d, Μέσος Όρος = %.2f\n",

sum,average);

}
```

Βλέπουμε ότι ο βρόχος for μάς δίνει την δυνατότητα για έναν απλό, εύκολο και ευθύ τρόπο χειρισμού των δεικτών του πίνακα. Η παρακάτω εντολή είναι πολύ βολική για την επεξεργασία των στοιχείων ενός πίνακα μεγέθους SIZE :

```
for (i=0; i<SIZE; i++)
```

Βρόχοι και Συναρτήσεις

Ακολουθεί ένα πρόγραμμα :

```
#include <stdio.h>

main()

{

double x, xpow;

double power(); /* δήλωση της συνάρτησης */

int n;

printf("Δώστε έναν αριθμό και μια ακέραια δύναμη ");

printf("Δώστε q για να σταματήσετε.");

while (scanf("%lf%d", &x, &n) == 2)

{

xpow=power(x,n); /* κλήση της συνάρτησης */

printf("Το %.3e εις την %d είναι %.3e \n",x,n, xpow);

}

}

double power(a,b) /* ορισμός της συνάρτησης */
```

double a;

int b;

{

double pow = 1;

int i;

for (i = 1; i <= b; i++)

*pow *= a;*

return pow; / επιστρέφει την τιμή της pow */*

}

Εδώ υπολογίζουμε το αποτέλεσμα της ύψωσης ενός αριθμού σε μια ακέραια δύναμη με τη χρήση μιας συνάρτησης. Η συνάρτηση δέχεται δύο τιμές και επιστρέφει μία. Χρησιμοποιούμε ένα όρισμα τύπου double και ένα όρισμα τύπου int, που καθορίζουν ποιος αριθμός θα υψωθεί σε ποια δύναμη.

Δηλώνουμε την συνάρτηση μέσα στην main() με τον τύπο της τιμής που επιστρέφει και με την χρήση της λέξης-κλειδί return δείχνουμε την τιμή που θα επιστρέψει η συνάρτηση. Τη συνάρτηση την καλούμε μέσα από το πρόγραμμα με την εξής εντολή :

xpow = power(x,n);.

Η Εντολή If

Η εντολή if καλείται *εντολή διακλάδωσης*, γιατί δημιουργεί ένα σημείο διασταύρωσης, από το οποίο το πρόγραμμα έχει να διαλέξει μεταξύ δύο δυνατών κατευθύνσεων που μπορεί να ακολουθήσει.

Η γενική μορφή (σύνταξη) της εντολής if είναι η εξής :

if*(έκφραση)*

πρόταση

Εάν η έκφραση είναι αληθής (δηλ. όχι μηδέν), τότε εκτελείται η πρόταση. Αλλιώς, αγνοείται. Η πρόταση μπορεί να είναι μια απλή πρόταση ή μια σύνθετη πρόταση.

Η C δίνει επίσης την δυνατότητα επιλογής μεταξύ δύο προτάσεων με την χρήση της δομής if else. Αν η έκφραση είναι αληθής, τότε εκτελείται ό,τι ακολουθεί την if, αλλιώς εκτελείται ό,τι ακολουθεί την else.

Η γενική μορφή (σύνταξη) της εντολής if else είναι η εξής :

if *(έκφραση)*

πρόταση1

else

πρόταση2

Ένα else συνδυάζεται με το πιο κοντινό του if, εκτός κι αν υπάρχουν αγκύλες, που δηλώνουν κάτι άλλο. Υπάρχει η δυνατότητα να χρησιμοποιήσουμε και την εντολήelse if, ως εξής :

if*(έκφραση1)*

πρόταση1

else if*(έκφραση2)*

πρόταση2

else

πρόταση3

Αν η *έκφραση1* είναι αληθής, τότε εκτελείται η *πρόταση1*. Αν η *έκφραση1* είναι ψευδής, αλλά η *έκφραση2* είναι αληθής, τότε εκτελείται η *πρόταση2*. Τέλος, αν και οι δύο εκφράσεις είναι ψευδείς, τότε εκτελείται η *πρόταση3*.

Οι Συναρτήσεις getchar() και putchar()

Ας δούμε τώρα κάτι άλλο. Η C έχει ένα ζευγάρι συναρτήσεων, ειδικά σχεδιασμένων για την είσοδο/έξοδο χαρακτήρων, τις getchar() και putchar(). Η συνάρτηση*getchar()*δεν έχει ορίσματα και επιστρέφει τον επόμενο χαρακτήρα που δίνεται ως δεδομένο, ως εξής :

ch = getchar();

Η συνάρτηση *putchar()* εκτυπώνει το όρισμά της, ως εξής :

putchar(ch);

Ακολουθούν δύο παραδείγματα :

#include <stdio.h>

```
#define SPACE ‘ ‘
main()
{
char ch;
ch = getchar(); /*διάβασεέναν χαρακτήρα*/
while(ch!= ‘\n’) /* αν δεν είναι το τέλος μιας γραμμής */
{
if(ch==SPACE) /* άφησε το κενό */
putchar(ch); /* αμετάβλητος χαρακτήρας */
else
putchar(ch+ 1); /* άλλαξε τους άλλους χαρακτήρες*/
ch=getchar(); /* επόμενος χαρακτήρας */
}
}
#include <stdio.h>
#define SPACE ‘ ‘
main()
{
char ch;
while ((ch = getchar()) != ‘\n’) /* αν δεν είναι το τέλος μιας γραμμής */
{
if(ch==SPACE) /* άφησε το κενό */
putchar(ch); /* αμετάβλητος χαρακτήρας */
else
putchar(ch+ 1); /* άλλαξε τους άλλους χαρακτήρες*/
}
}
```

Τα προγράμματα αυτά επαναλαμβάνουν μια γραμμή εισόδου, αντικαθιστώντας κάθε χαρακτήρα εκτός του κενού, με τον επόμενό του ASCII χαρακτήρα. Τα κενά παραμένουν αμετάβλητα. Ένα αποτέλεσμα από τη χρήση των προγραμμάτων μπορεί να είναι το εξής :

FLORINA PER SEMPRE.

GMPSJOB QFS TFNQSF/

Η ευελιξία της C μάς επιτρέπει να συνδυάσουμε την ανάγνωση και τον έλεγχο των δεδομένων εισόδου σε μία μόνο έκφραση. Και αυτό το βλέπουμε στο δεύτερο πρόγραμμα και στην εντολή *while ((ch=getchar())!='\n')*, όπου έχουμε σε μια έκφραση την καταχώρηση μιας τιμής στη ch και σύγκριση αυτής της τιμής με τον χαρακτήρα νέας γραμμής \n. Αν δεν βάζαμε τις παρενθέσεις, τότε η πρώτη έκφραση που θα εκτελείτο, θα ήταν η *getchar()!='\n'*.

Οι Λογικοί Τελεστές

Στο παρακάτω παράδειγμα συνδυάζουμε τρεις σχεσιακές εκφράσεις με τον λογικό τελεστή *&&*, που είναι το γνωστό μας AND στην C. Το πρόγραμμα μετρά τους *μη-λευκούς χαρακτήρες*, δηλ. τους χαρακτήρες που δεν είναι κενό, enter και tab.

```
#include <stdio.h>

#define PERIOD '.'

main()

{

int ch;

int charcount = 0;

while ((ch = getchar()) != PERIOD)

if (ch != ' ' && ch != '\n' && ch != '\t')

charcount++;

printf("Υπάρχουν %d μη-λευκοί χαρακτήρες. \n"charcount);

}
```

Οι λογικοί τελεστές έχουν μικρότερη προτεραιότητα από τους σχεσιακούς τελεστές και υπάρχουν τρεις απ' αυτούς στην C :

<u>*Τελεστής Σημασία*</u>

&& And (και)

|| *Or (ή)*

! Not (όχι)

Ο τελεστής ! έχει πολύ μεγάλη προτεραιότητα, μεγαλύτερη και από εκείνη του πολλαπλασιασμού, ίδια με την προτεραιότητα των σχεσιακών τελεστών και μικρότερη από εκείνη των παρενθέσεων.Ο τελεστής && έχει μεγαλύτερη προτεραιότητα από τον ||, αλλά και οι δύο βρίσκονται κάτω από τους σχεσιακούς τελεστές και πάνω από τους τελεστές καταχώρησης. Ακόμη, οι λογικές εκφράσεις στην C υπολογίζονται από αριστερά προς τα δεξιά.

Ο Τελεστής υπό Συνθήκη ?

Ένας άλλος τρόπος για να εκφράσουμε την πρόταση if else ονομάζεται *έκφραση υπό συνθήκη* και χρησιμοποιεί τον τελεστή υπό συνθήκη ? : που έχει τρεις τελεστέους. Το παρακάτω παράδειγμα βρίσκει την απόλυτη τιμή ενός αριθμού :

x = (y<0) ? -y : y;

Η πρόταση λέει τα εξής *«αν το y είναι μικρότερο από το 0, τότε x=-y, αλλιώς x=y»*.

Η γενική μορφή (σύνταξη) της έκφρασης υπό συνθήκη είναι η εξής :

έκφραση1 ? έκφραση2 : έκφραση3

Αν η *έκφραση1* είναι αληθής (δηλ. όχι μηδέν), τότε ολόκληρη η έκφραση υπό συνθήκη έχει την τιμή της *έκφρασης2*, αν όμως η *έκφραση1* είναι ψευδής (μηδέν), τότε ολόκληρη η έκφραση υπό συνθήκη έχει την τιμή της *έκφρασης3*.

Πώς βρίσκουμε τον μέγιστο από δύο αριθμούς :

max = (a>b) ? a : b

Ακολουθούν παραδείγματα :

(5>2) ? 1 : 2 έχει τιμή 1

(3>5) ? 1 : 2 έχει τιμή 2

(a>b) ?a:b έχει την τιμή του μεγαλύτερου μεταξύ των a και b

Η Εντολή Switch

Η εντολή switch είναι η πλέον κατάλληλη όταν έχουμε να επιλέξουμε από πολλές εναλλακτικές περιπτώσεις, όπου η χρήση διαδοχικών if else δυσκολεύει πολύ τον κώδικα του προγράμματος.

Ακολουθεί ένα πρόγραμμα, στο οποίο διαβάζουμε ένα γράμμα και μετά μας δίνει το όνομα μιας πόλης που ν' αρχίζει μ' αυτό το γράμμα.

```
#include <stdio.h>

main()

{

char ch;

printf("Δώστε ένα γράμμα του αλφαβήτου.");

printf("Πατήστε # για να τελειώσει το πρόγραμμα : \n");

while ((ch = getchar()) != '#')

{

if (ch >= 'a' && ch <= 'z') /*δέχεται μόνο */

 switch (ch) /*μικρά γράμματα*/

{

case 'a' :

printf("Αθήνα\n");

break;

case 'b' :

printf("Βόλος\n");

break;

case 'c' :

printf("Γρεβενά\n");

break;

case 'd' :

printf("Δράμα\n");

break;

case 'e' :

printf("Έδεσσα \n");

break;
```

default :

printf("Δεν υπάρχει πόλη\n");

*} /*τέλοςτηςswitch */*

else

printf("Αναγνωρίζω μόνο μικρά γράμματα \n");

while (getchar() != '\n')

*; /*παρέλειψε το υπόλοιπο της γραμμής εισόδου */*

printf("Παρακαλώ, δώστε ένα άλλο γράμμα ή #.\n");

} / τέλος βρόχου while */*

}

Η εντολή *switch* δουλεύει ως εξής :

Πρώτα, υπολογίζεται η έκφραση μέσα στις παρενθέσεις, που ακολουθεί την switch. Μετά, το πρόγραμμα σαρώνει τον κατάλογο με τις ετικέττες *case* μέχρι να βρει μια που να ταιριάζει μ' αυτήν την τιμή. Τότε, το πρόγραμμα πηγαίνει σ' αυτήν την γραμμή και εκτελεί τις εντολές που περιέχονται σ' αυτήν. Αν δεν ταιριάζει με καμία επιλογή, τότε το πρόγραμμα πηγαίνει στην γραμμή με την ετικέττα *default*, αν υπάρχει αυτή βέβαια. Αν δεν υπάρχει η επιλογή default, τότε το πρόγραμμα συνεχίζει κανονικά στην επόμενη εντολή μετά την switch.

Αν δεν υπήρχε η πρόταση *break*, τότε το πρόγραμμα θα εκτελούσε κάθε πρόταση από την ετικέττα ταύτισης μέχρι και το τέλος της πρότασης switch. Δεν μπορούμε να χρησιμοποιήσουμε μια μεταβλητή για ετικέττα στην πρόταση switch, αλλά μόνο εκφράσεις σταθερών τύπου int ή char.

Η δομή της εντολής switch είναι η εξής :

***switch** (ακέραια έκφραση)*

{

***case**(σταθερά 1 :*

προτάσεις; (προεραιτικά)

***case**(σταθερά 2 :*

προτάσεις; (προεραιτικά)

...

***default**: (προεραιτικά)*

προτάσεις; (προεραιτικά)

}

Η Εντολή Break

Η εντολή αυτή μπορεί να χρησιμοποιηθεί με τη switch, όπως είδαμε νωρίτερα, αλλά και με τις υπόλοιπες τρεις δομές βρόχου και έχει ως αποτέλεσμα την διακοπή της εκτέλεσης των εντολών switch, for, while και do while και την μετάβαση στο επόμενο στάδιο του προγράμματος. Ακολουθεί ένα κομμάτι προγράμματος που σταματά έναν βρόχο όταν διαβαστεί είτε ένα πλήκτρο enter ή ένα πλήκτρο tab :

while ((ch=getchar()) != '\n')

{

if (ch == '\t')

break;

putchar(ch);

}

Το παραπάνω κομμάτι προγράμματος θα μπορούσε να δουλέψει και ως εξής :

while ((ch=getchar()) != '\n' && ch !='\t')

putchar(ch);

Η Εντολή Continue

Αυτή η εντολή μπορεί να χρησιμοποιηθεί και στα τρία είδη βρόχων, αλλά όχι με την πρόταση switch. Και η continue διακόπτει την ροή ενός προγράμματος, αλλά αντί να τερματίζει ολόκληρο τον βρόχο, έχει ως αποτέλεσμα την μη εκτέλεση του υπόλοιπου τμήματος του βρόχου και την επανάληψή του από την αρχή.

Σ' έναν βρόχο while ή for, ξαναρχίζει ο επόμενος κύκλος του βρόχου, ενώ σ' έναν βρόχο do while ελέγχεται η συνθήκη εξόδου και μετά, αν χρειαστεί, αρχίζει ο επόμενος κύκλος του βρόχου. Τα δύο παραπάνω κομμάτια προγράμματος θα μπορούσαν να γραφούν τώρα ως εξής :

while ((ch=getchar()) != '\n')

{

if (ch == '\t')

continue;

putchar(ch);

}

Δηλαδή ο βρόχος σταματάει τώρα μόνο με το πλήκτρο enter και όχι και με το tab. Το παραπάνω κομμάτι προγράμματος θα μπορούσε να δουλέψει κι έτσι :

while ((ch=getchar()) !='\n')

if (ch != '\t')

putchar(ch);

Αποφυγή Χαρακτήρων Κατά το Διάβασμα

Επειδή υπάρχει πάντα ο κίνδυνος ο χαρακτήρας νέας γραμμής <enter> να διαβαστεί ως ξεχωριστός χαρακτήρας και να δημιουργηθούν έτσι προβλήματα σε διαλογικά προγράμματα, μια λύση στο πρόβλημα αυτό είναι με τη χρήση της εντολής while :

while (getchar() != '\n') / αγνοεί το υπόλοιπο της γραμμής εισόδου */*

Πρέπει να έχουμε υπόψη μας ότι η getchar() διαβάζει κάθε χαρακτήρα, συμπεριλαμβανομένων και των χαρακτήρων κενού διαστήματος (space), στηλοθέτησης (tab) και νέας γραμμής (enter), ενώ η scanf(), όταν διαβάζει αριθμούς, αγνοεί αυτούς τους χαρακτήρες.Μόνο όταν διαβάζουμε χαρακτήρες χρησιμοποιώντας τον προσδιοριστή %c, η scanf() συμπεριφέρεται όπως η getchar().

Για να αποφύγουμε τυχόν προβλήματα με το διάβασμα των χαρακτήρων που προαναφέραμε, μπορούμε να χρησιμοποιήσουμε την εξής εντολή if μέσα στο πρόγραμμά μας :

if (ch != '\n' && ch != ' ' && ch !='\t')

για να αποφύγουμε έτσι αυτούς τους χαρακτήρες.

Προσπέλαση Αρχείων.

Έως τώρα σε ένα πρόγραμμα έχουμε μάθει να εισάγουμε δεδομένα από το πληκτρολόγιο χρησιμοποιώντας την συνάρτηση scanf() και να εκτυπώνουμε δεδομένα στην οθόνη του υπολογιστή μας με την συνάρτηση printf(). Το πληκτρολόγιο για το πρόγραμμά μας αποτελεί την καθιερωμένη

είσοδο γιαυτό και ονομάζεται stdin (standard input), ενώ η οθόνη την καθιερωμένη έξοδο και ονομάζεται stdout (standard output), Η χρήση του πληκτρολογίου και της οθόνης ως είσοδος□έξοδος στα προγράμματά μας δεν είναι πάντοτε η ενδεδειγμένη. Σκεφθείτε εάν έχετε να εισάγετε μερικές εκατοντάδες αριθμούς στο πρόγραμμά σας, αυτό είναι πρακτικά αδύνατο να γίνει από το πληκτρολόγιο. Το ίδιο ισχύει εάν ένα πρόγραμμα παράγει πολλές γραμμές στην έξοδό του. Σε αυτή την περίπτωση η λύση είναι να χρησιμοποιήσουμε κατάλληλα Αρχεία Δεδομένων, τα οποία το πρόγραμμα χρησιμοποιεί ως είσοδο□έξοδο.

Σε αυτή την παράγραφο θα ασχοληθούμε με το πώς μέσα από ένα πρόγραμμα μπορούμε να δημιουργήσουμε ένα νέο αρχείο και να γράψουμε μέσα σε αυτό δεδομένα.Ας υποθέσουμε πως θέλουμε να αναπτύξουμε ένα πρόγραμμα το οποίο να δημιουργήσει ένα αρχείο δεδομένων και να γράψει σε αυτό τα δεδομένα που περιέχει μια μεταβλητή x=3.14159. Η λύση φαίνεται παρακάτω:

```
#include<stdio.h>

int main(void){

double x;

/* Δήλωση μεταβλητής x */

FILE *fp;

/* Δήλωση δείκτη αρχείου με όνομα fp */

x=3.14159;

/* Αρχικοποίηση μεταβλητής x */

fp=fopen(“arxeio.data”,”w”);

/* Δημιουργία αρχείου arxeio.data για εγγραφή */

fprintf(fp,”%f”,x);

/* Εγγραφή των δεδομένων στο αρχείο */

fclose(fp);

/* Κλείσιμο του αρχείου */

return 0;

}
```

Για την δημιουργία και την εγγραφή δεδομένων σε αρχείο απαιτούνται τα εξής βήματα:

Το πρώτο βήμα είναι η δήλωση ενός δείκτη αρχείου. Αυτό επιτυγχάνεται με την γραμμή:

FILE *fp;

Ο δείκτης fp (μπορούμε να χρησιμοποιήσουμε εδώ όποιο όνομα θέλουμε σύμφωνα με τους κανόνες ονοματολογίας των μεταβλητών) είναι τύπου FILE, και μπορεί να διαχειρίζεται ένα αρχείο. Η δημιουργία του αρχείου γίνεται στην γραμμή:

fp=fopen("arxeio.data","w");

Εδώ εμπλέκεται η συνάρτηση fopen() η οποία παίρνει δύο ορίσματα: Το πρώτο όρισμα είναι το όνομα του αρχείου που θέλουμε να δημιουργήσουμε, στο συγκεκριμένο παράδειγμα το όνομα που επιλέξαμε είναι το arxeio.data. Το δεύτερο όρισμα προσδιορίζει τι ακριβώς θέλουμε να κάνουμε με το συγκεκριμένο αρχείο, στο συγκεκριμένο παράδειγμα πρέπει να χρησιμοποιήσουμε ως δεύτερο όρισμα το "w" (αρχικό γράμμα της λέξης write) το οποίο σημαίνει πως δημιουργώ ένα νέο αρχείο το οποίο προορίζεται για εγγραφή δεδομένων. Προσοχή! Το όρισμα w θα δημιουργήσει στον σκληρό δίσκο του υπολογιστή το αρχείο με όνομα arxeio.data. Εάν ήδη υπάρχει αρχείο με αυτό το όνομα τα δεδομένα μέσα σε αυτό θα χαθούν. Κατά συνέπεια πρέπει να είμαστε πολύ προσεκτικοί. Για την εγγραφή δεδομένων μέσα στο αρχείο που δημιουργήσαμε χρησιμοποιούμε την συνάρτηση fprintf() όπως στη γραμμή:

fprintf(fp,"%f",x);

Η σύνταξη της συνάρτησης fprintf() είναι ακριβώς η ίδια με αυτή της printf() εκτός του ότι παίρνει ένα επί πλέον όρισμα (το πρώτο) το οποίο είναι ο δείκτης του αρχείου. Τέλος όταν έχουμε τελειώσει πλήρως με το αρχείο μας, πρέπει να το κλείσουμε με τη χρήση της συνάρτησης fclose() όπως στη γραμμή:

fclose(fp);

Άνοιγμα αρχείου και διάβασμα δεδομένων από αυτό. Σε αυτή την παράγραφο θα ασχοληθούμε με το πώς μέσα από ένα πρόγραμμα μπορούμε να "ανοίξουμε" ένα αρχείο το οποίο προϋπάρχει στον σκληρό μας δίσκο αποθηκεμένο και να διαβάσουμε

από αυτό δεδομένα. Ας υποθέσουμε πως στον σκληρό δίσκο του υπολογιστή μας προϋπάρχει το αρχείο δεδομένων με όνομα dedomena.txt και πως μέσα σε αυτό εμπεριέχεται ο αριθμός 7.689. Θέλουμε να αναπτύξουμε ένα πρόγραμμα το οποίο να "ανοίξει" το συγκεκριμένο αρχείο, να διαβάσει τον αριθμό, να τον αποθηκεύσει σε μια κατάλληλη μεταβλητή και να τον τυπώσει στην οθόνη του υπολογιστή μας.

```
#include<stdio.h>

int main(void){

double y;

/* Δήλωση μεταβλητής y */

FILE *pp;

/* Δήλωση δείκτη αρχείου με όνομα pp*/

pp=fopen("dedomena.txt ","r");

/* Άνοιγμα αρχείου dedomena.txt για διάβασμα */

/* Διάβασμα των δεδομένων και αποθήκευση στην y */

fscanf(pp,"%lf",&y);

printf("y=%f \n",y);

/* Εκτύπωση στην οθόνη της μεταβλητής y */

fclose(pp);

/* Κλείσιμο του αρχείου */

return 0;

}
```

Για το άνοιγμα του αρχείου dedomena.txt και το διάβασμα απαιτούνται τα εξής βήματα:

Το πρώτο βήμα είναι η δήλωση ενός δείκτη αρχείου. Αυτό επιτυγχάνεται με την γραμμή:

```
FILE *pp;
```

Ο δείκτης pp (μπορούμε να χρησιμοποιήσουμε εδώ όποιο όνομα θέλουμε σύμφωνα με τους κανόνες ονοματολογίας των μεταβλητών) είναι τύπου FILE,

και μπορεί να διαχειρίζεται ένα αρχείο. Το “άνοιγμα” του αρχείου dedomena.txt γίνεται στην γραμμή:

pp=fopen(“dedomena.txt ”,”r”);

Εδώ εμπλέκεται η συνάρτηση fopen() η οποία παίρνει δύο ορίσματα: Το πρώτο όρισμα είναι το όνομα του αρχείου από το οποίο θέλουμε να διαβάσουμε, στο συγκεκριμένο παράδειγμα το όνομα του αρχείου το οποίο προϋπάρχει στον σκληρό δίσκο του υπολογιστή μας είναι το dedomena.txt. Το δεύτερο όρισμα προσδιορίζει τι ακριβώς θέλουμε να κάνουμε με το συγκεκριμένο αρχείο, στο συγκεκριμένο παράδειγμα πρέπει να χρησιμοποιήσουμε ως δεύτερο όρισμα το “r” (αρχικό γράμμα της λέξης read) το οποίο σημαίνει πως το αρχείο προορίζεται για ανάγνωση δεδομένων. Προσοχή! Εάν το αρχείο με όνομα dedomena.txt δεν υπάρχει το πρόγραμμα θα τερματιστεί παράγοντας μήνυμα λάθους. Για το διάβασμα δεδομένων από το αρχείο dedomena.txt χρησιμοποιούμε την συνάρτηση fscanf() όπως στη γραμμή:

fscanf(pp,”%lf”,&y);

Η σύνταξη της συνάρτησης fscanf() είναι ακριβώς η ίδια με αυτή της scanf() εκτός του ότι παίρνει ένα επί πλέον όρισμα (το πρώτο) το οποίο είναι ο δείκτης του αρχείου. Τέλος όταν έχουμε τελειώσει πλήρως με το αρχείο μας πρέπει να το κλείσουμε με τη χρήση της συνάρτησης fclose() όπως στη γραμμή:

fclose(pp);

Εδώ συνοψίζοντας σχετικά με την συνάρτηση fopen(), το δεύτερο όρισμα έχει την ακόλουθη έννοια:

w : Δημιουργία νέου αρχείου προς εγγραφή δεδομένων.

r : “Άνοιγμα” αρχείου το οποίο προϋπάρχει προς ανάγνωση δεδομένων.

a: “Άνοιγμα” αρχείου το οποίο προϋπάρχει προς πρόσθεση νέων δεδομένων μετά το τέλος του. Οι συναρτήσεις putc() και getc(),γενικά η είσοδος□έξοδος κειμένου στα προγράμματα είναι πολύ σημαντική και γιαυτό τον λόγο στη C υπάρχουν ειδικές συναρτήσεις εισόδου□εξόδου που αφορούν αποκλειστικά χαρακτήρες. Οι συναρτήσεις αυτές είναι οι getc() και putc(). Οι δύο αυτές συναρτήσεις μπορούν να χρησιμοποιηθούν είτε για είσοδο από το πληκτρολόγιο είτε για έξοδο στην οθόνη του υπολογιστή μας Η σύνταξη των συναρτήσεων getc() και putc() εικονίζεται στις παρακάτω γραμμές τόσο για την

είσοδο των χαρακτήρων από το πληκτρολόγιο όσο και για την έξοδο στην οθόνη. Για να γίνει κατανοητή η χρήση τους εικονίζεται δίπλα και ο αντίστοιχος κώδικας με τις συναρτήσεις scanf() και printf().

Είσοδος χαρακτήρων από το πληκτρολόγιο Συνάρτηση getc() Συνάρτηση scanf()

```
char c;      char c;
scanf("%c",&c);
c=getc(stdin);
```

Συνάρτηση putc() Συνάρτηση printf()

```
char c;      char c;
c='k';
c='k';     printf("%c",c);
putc(c,stdout);
```

Έξοδος χαρακτήρων στην οθόνη

Οι συναρτήσεις getc() και putc() μπορούν να ανακατευθύνουν χαρακτήρες και σε κατάλληλα αρχεία εισόδου□εξόδου, αρκεί στην θέση των stdin και stdout να χρησιμοποιήσουμε τους αντίστοιχους δείκτες αρχείων. Ακολουθεί η σύνταξη των συναρτήσεων getc() και putc() τόσο για την είσοδο χαρακτήρων από αρχείο δεδομένων όσο και για την έξοδο.

Είσοδος χαρακτήρων από αρχείο δεδομένων

```
char c;
/* Δήλωση μεταβλητής τύπου char */
FILE *fp;
/* Δήλωση δείκτη αρχείου με όνομα fp*/
fp=fopen("dedomena.txt","r");
 /* Άνοιγμα αρχείου dedomena.txt για διάβασμα */
c=getc(fp);
/* Διάβασμα ενός χαρακτήρα από το αρχείο και αποθήκευση στην c */
fclose(fp);
```

```
/* Κλείσιμο του αρχείου */

Έξοδος χαρακτήρων σε αρχείο δεδομένων

char  c;

/* Δήλωση μεταβλητής τύπου char */

FILE *fp;

/* Δήλωση δείκτη αρχείου με όνομα fp*/

c='k';

/* Αρχικοποίηση της μεταβλητής c */

fp=fopen("arxeio.data","w");

/* Δημιουργία αρχείου arxeio.data για εγγραφή */

putc(c, fp);

/* Γράψιμο του χαρακτήρα στο αρχείο */

fclose(fp);

/* Κλείσιμο του αρχείου */
```

Σημειώνουμε εδώ πως στην περίπτωση ανακατεύθυνσης χαρακτήρων σε αρχεία εισόδου□εξόδου η C είναι εφοδιασμένη και με τις συναρτήσεις fgetc() και fputc() οι οποίες είναι ακριβώς ισοδύναμες με τις getc() και putc().

Τέλος κλείνοντας αυτή την παράγραφο θέλουμε να επισημάνουμε κάτι πολύ χρήσιμο για τα προγράμματά μας. Όταν διαβάζουμε έναν□έναν του χαρακτήρες ενός αρχείου, για να αντιληφθούμε πως έχουμε φτάσει στο τέλος του, πρέπει να ελέγχουμε για έναν ειδικό χαρακτήρα οποίος σημαίνει το τέλος ενός αρχείου. Ο χαρακτήρας αυτός είναι ο EOF (από τα αρχικά των αγγλικών λέξεων End Of File). Για παράδειγμα στις επόμενες γραμμές δίνουμε έναν εύκολο τρόπο για να διαβάσουμε όλους τους χαρακτήρες ενός αρχείου έως το τέλος του:

```
 char  c;

/* Δήλωση μεταβλητής τύπου char */

FILE *fp;

/* Δήλωση δείκτη αρχείου με όνομα fp*/

fp=fopen("dedomena.txt","r");
```

```
/* Άνοιγμα αρχείου dedomena.txt για διάβασμα */
while((c=getc(fp))!=EOF){
    .......
     ......
}
fclose(fp);
/* Κλείσιμο του αρχείου */
```

Στην γραμμή: while((c=getc(fp))!=EOF){ .... } διαβάζουμε πρώτα έναν□έναν τους χαρακτήρες του κειμένου. Η εντολή while εκτελείται όσο ο χαρακτήρας που διαβάζουμε από το αρχείο δεν είναι ίσος με τον EOF.