Δυναμικές Δομές Δεδομένων

Μια μικρή εισαγωγή στις γραμμικές λίστες σε C

Απο τον AtticaDreamer

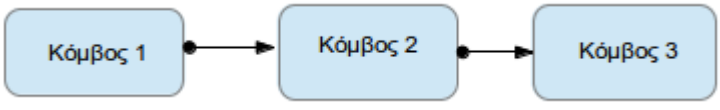
Καλως ορίσατε στο tutorial του 101projects.eu για δημιουργία συνδεδεμένης λίστας σε γλώσσα προγραμματισμού C . Για αυτό το tutorial θα δημιουργήσουμε και θα κατανοήσουμε μια συνδεδεμένη λίστα,σε όλες τις τις παραλαγές,τελείως απο το μηδέν,χρησιμοποιώντας μόνο δικό μας κώδικα! Στο τέλος αυτού του tutorial, θα γνωρίζετε πλήρως πώς λειτουργεί αυτή η *δυναμική δομή δεδομένων* καθώς και να την υλοποιείτε σε γλώσσα προγραμματισμού C .Σε αυτό το tutorial όλα θα εξηγηθούν επαρκώς ,συνεπώς ακόμα και για τον πιο αρχάριο προγραμματιστή,η εκμάθηση αυτής της δομής θα είναι παιχνιδάκι,Όμως για την ευκολότερη κατανοήση του tutorial αυτού,συνίσταται η γνώση ΤΟΥΛΆΧΙΣΤΟΝ των παρακάτω δυνατοτήτων της γλώσσας προγραμματισμού C (Η της γλώσσας της αρισκείας σας)

- Δομές (Structs)
- Δείκτες(Pointers) καθώς και την αριθμητική τους
- Δυναμική διαχείριση μνήμης με συναρτήσεις malloc() και free()
- Δημιουργία Συναρτήσεων τύπου-δείκτη με παραμέτρους δείκτες

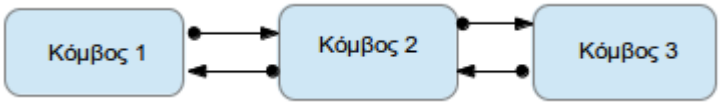
Λίγη Θεωρία : Τί είναι δομή δεδομένων;

Δομή δεδομένων ονομάζουμε τον τρόπο (την αρχιτεκτονική) με τον οποίο οργανώνουμε ένα σύνολο δεδομένων στην μνήμη.Υπάρχουν αρκετές δομές δεδομένων οι οποίες είναι αποδότικες ανάλογα με την περίπτωση στην οποία θα χρησιμοποιήθουν ή το πρόβλημα που πρέπει να λύσουν! Ας δούμε τις βασικές δομές δεδομένων, με μια μικρή περιληπτική εισαγωγή για κάθε μία

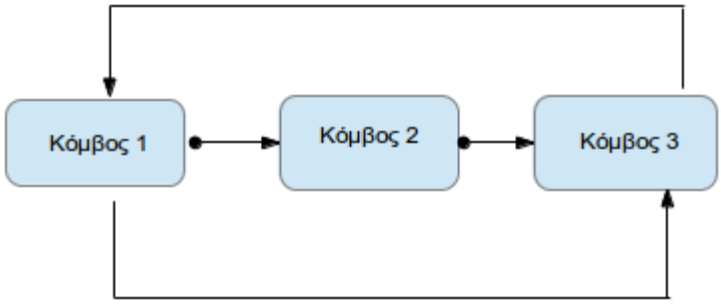
- Γραμμικές Λίστες
  Οι γραμμικές λίστες είναι η απλούστερη δομή δεδομένων που θα μπορούσε να υπάρξει , έπειτα απο τον πίνακα. Στην δομή αυτή , κάθε κόμβος συνδεέται σειριακά με τον επόμενο του. Οι γραμμικές λίστες διακρίνονται στις εξής κατηγορίες
  1. *Απλά συνδεδεμένες γραμμικές λίστες.* Όπου κάθε κόμβος απλά συνδέεται με τον επόμενο του



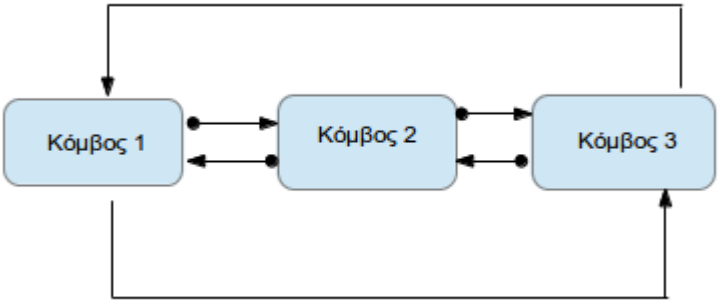
2. *Διπλά συνδεδεμένες γραμμικές λίστες(η διπλοδεσμικές λίστες)* , Όπου κάθε κόμβος περιέχει σύνδεση για τον επόμενο ***και*** τον προηγούμενο κόμβο του



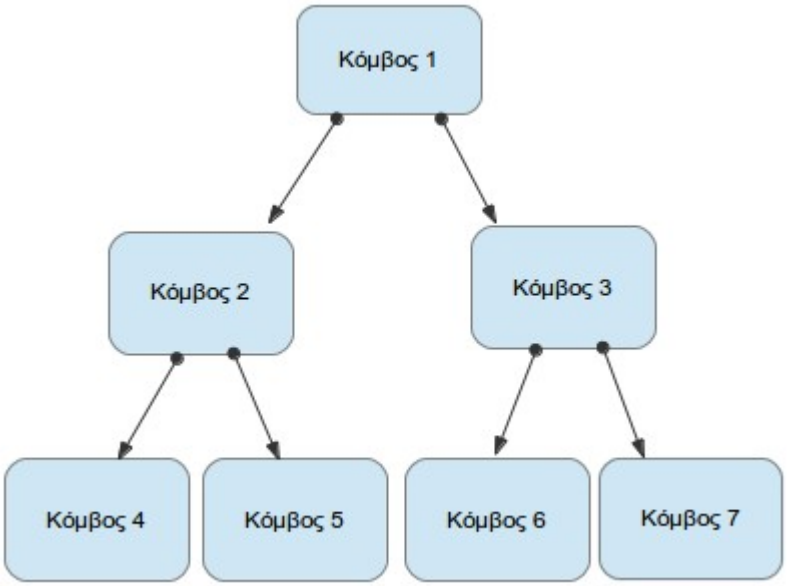
3. *Κυκλικά συνδεδεμένες γραμμικές λίστες* , όπου κάθε κόμβος συνδέεται σειριακά με τον επόμενο του , ο τελευταίος κόμβος περιέχει σύνδεση για τον πρώτο κόμβο.Ο πρώτος κόμβος με την σειρά του, περιέχει σύνδεση στον τελευταίο κόμβο.



4. *Κυκλικα-Διπλά συνδεδεμένες λίστες*, όπου κάθε κόμβος έχει σύνδεση με τον επόμενο και τον προηγούμενο κόμβο του , ο τελευταίος κόμβος περιέχει σύνδεση για τον πρώτο κόμβο.Ο πρώτος κόμβος με την σειρά του, περιέχει σύνδεση στον τελευταίο κόμβο.



- Δυαδικό Δέντρο(Binary Tree)
  Το δυαδικό δέντρο είναι μια δομή δεδομένων με δενδροειδή δομή.Κάθε κόμβος δεδομένων στην μνήμη , συνδεέται με δύο κόμβους χαμηλότερου επιπέδου , και μόνο με έναν κόμβου υψηλότερου επιπέδου! ,η σύνδεση αυτή, είναι αναλόγως τα κριτίρια που εμείς θέτουμε.Τα δυαδικά δέντρα θα τα δούμε σε επόμενα tutorials !



- Γράφημα(Graph)
  Αυτή είναι μια απο τις πιο πολύπλοκες δομές δεδομένων , η οποία έχει την εξής ιδιότητα. Κάθε κόμβος έχει ένα δικό του σύνολο απο κόμβους – γείτονες μεγέθους n , επίσης υπάρχει ένα σύνολο στο οποίο αποθηκέυεται το κόστος της κάθε σύνδεσης. . Το γεγονός οτι ο γράφος δέν έχει κάπια συγκεκριμένη φόρμα , τον κάνει ικανό για χαρτογραφήσεις και αναζητήσεις !. Θα αναφερθούμε στους γράφους σε μετέπειτα tutorials !

Και γιατί δεν χρησιμοποιούμε πίνακες;

Ο πίνακας είναι μια αρκετά απλή δομή δεδομένων με αρκετά μειονεκτήματα.Το μεγαλύτερο απο αυτά είναι οτι δεν λειτουργεί *Δυναμικά.*

Τι εννοούμε με τον όρο Δυναμικά;

Όταν δηλώνουμε έναν πίνακα,καθορίζουμε απο την αρχή το μέγεθος του,χωρίς να μπορούμε να το μεταβάλλουμε στην συνέχεια .Ο πίνακας αυτός θα παραμείνει σταθερός καθ όλη την διάρκεια του προγράμματος.Αν εμείς ομώς δεν μπορούμε να προβλέψουμε το μέγεθος των δεδομένων που θα λαμβάνουμε; Τι συμβαίνει σε αυτή την λεπτή (αλλα καθόλου σπάνια) περίπτωση; Εκεί μας δίνουν την λύση οι δυναμικές δομές δεδομένων , στις οποίες το μέγεθος τους αυξομοιώνεται δυναμικά μέσα στην μνήμη διατηρώντας ανέπαφα τυχόν αλλα δεδομένα προγράμμάτων που βρίσκονται αποθηκευμένα σε αυτή.Την προστασία των άλλων προγραμμάτων απο αλλοίωση δεδομένων την παρέχει το σύστημα *δυναμικής διαχείρισης μνήμης* της γλώσσας C

Και χρειαζόμαστε 30 (και μια...) σελίδες Tutorial για να μάθουμε μια “απλή” δομή δεδομένων,όπως είναι οι γραμμικές λίστες;

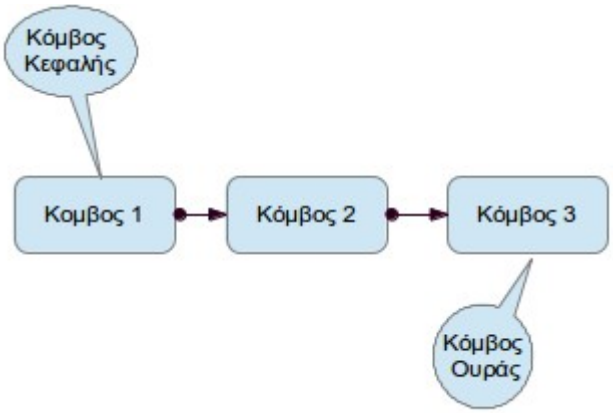
Λοιπόν,η απάντηση είναι ,“Ανάλογα που στοχεύεις”.Αν θές να γίνεις σωστός προγραμματιστής , είναι προφανές οτι πρέπει να ξέρεις πως λειτουργούν τα πράγματα , σε πιο Low-level επίπεδο ,πριν περάσεις ,στις έτοιμες λύσεις .Αν απλά θές να φιάξεις ενα πρόγραμμα,και δεν σε ενδιαφέρει,τότε δεν χρειάζεται να διαβάσεις τόσο πολυ . Αλλα θυμίσου , στην αγορά εργασίας , πλεονέκτημα θα έχει αυτός που διάβασε πιο πολύ ;)

Πως ξεκινάμε?

Δεν χρειαζόμαστε πραγματικά τίποτε αλλο απο έναν απλό IDE της C για να ξεκινήσουμε να γράφουμε κώδικα.!.Εγώ παραδειγματικά θα χρησιμοποιήσω Sublime Text Editor και gcc compiler (έκδοση 4.6.3 Πάνω σε λειτουργικό σύστημα Elementary OS 64bit .)

Βουτία στα βαθία: ***Κόμβοι και συνδέσεις…..***

Έχουμε ακούσει άπειρες φορές την λέξη “Κόμβος” , χωρίς να έχουμε αναλύσει εις βάθος τι ακριβώς εννοούμε!Γενικότερα οι δομές δεδομένων,χρησιμοποιούν κόμβούς για να μπορέσουν να αποθηκεύσουν τα δεδομένα που θέλουν,καθώς και για να μπορούν να τα συνδέσουν μεταξύ τους ! ,Ο πρώτος κόμβος της δομής μας ονομάζεται head node (ή κόμβος κεφαλής) και ο τελευταίος ονομάζεται Tail node (Κόμβος Ουράς) .

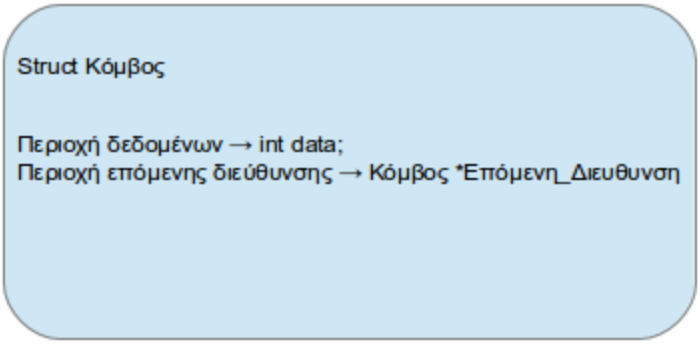


Τι ακριβώς είναι αυτός ο <<κόμβος>>;

Ο κόμβος δεν είναι τίποτε παραπάνω απο μια μεταβλητή δομής (struct) η οποία περιέχει δύο πεδία . Το πρώτο πεδίο το ονομάζουμε *περιοχή δεδομένων του κόμβου* και αποθηκεύει τα δεδομένα του κόμβου ,ενώ το δεύτερο το ονομάζουμε *περιοχη επόμενης διεύθυνσης* και εκεί αποθηκεύουμε την διεύθυνση του επόμενου κόμβου.Την μορφή καθώς και την σύνταξη ενός κόμβου θα την δούμε αργότερα

Με την ματία ενός προγραμματιστή….

Η γενική μορφή των κόμβων που θα απαρτίζουν την δομή μας θα είναι κάπως έτσι



Σημείωση!

Δέν πρέπει να συνχέεται η δομή (Struct) με την δομή δεδομένων (Data Structure) . Δομή (Struct) ονομάζουμε μια συλλογή απο μεταβλητές και πίνακες οποιοδήποτε τύπου (Int,float,bool,char,string κ.α),ενώ δομή δεδομένων ονομάζουμε τον τρόπο οργάνωσης των δεδομένων μας στην μνήμη.

Για κάπιον που γνωρίζει C , Αυτή η γενική μορφή κόμβου θα πρέπει να του θυμίζει πολλα! Το σχήμα αυτό,περιγράφει μια απλη δομή της C η οποία γραφοντάς την με κώδικα θα έμοιαζε κάπως έτσι….

```
1  #include <stdio.h>
2  struct NewNode
3  {
4      int data;
5      struct NewNode *ConnectionAddress;
6
7
8  };
```

Η δομή αυτή θα απαρτίζει την δομή δεδομένων μας, παίζοντας τον ρόλο του κόμβου .Γνωρίζοντας τα βασικά των δομών και των δεικτών, δεν θα πρέπει ο παραπάνω κώδικας να είναι δύσκολος την κατανόηση,παρόλα αυτά ,θα εξηγήσουμε γραμμή – γραμμή τι ακριβώς κάναμε!

*Σημείωση : Η δομή του κόμβου διαφέρει ανάλογα με τον τύπο γραμμικής λίστας που χρησιμοποιούμε! Τις διαφορές και τις αλλαγές θα τις σχολιάσουμε αργότερα

Εξηγώντας γραμμη-γραμμή…

Γραμμή 2: Δηλώνεται η δομή των κόμβων με το όνομα αναφοράς NewNode
Γραμμή 4: Δηλώνεται οτι η δομή θα περιέχει μια μεταβλητή τύπου integer με όνομα data. Αυτή η μεταβλητή θα περιέχει τα δεδομένα του κάθε κόμβου(στην περίπτωση μας <<δεδομένα>> είναι ένας ακέραιος αριθμός). Αυτή την μεταβλητή είναι η “Περιοχή Δεδομένων του κόμβου “
Γραμμή 5: Δηλώνεται οτι η δομή θα περιέχει μια μεταβλητή τύπου NewNode(τύπου της ίδιας της δομής) η οπόια θα είναι δείκτης(Δηλαδή θα αποθηκεύει διευθύνσεις) με όνομα ConnectionAddress Με λίγα λόγια αυτή η μεταβλητή θα μπορεί να αποθηκεύει διευθύνσεις τύπου NewNode.Αυτή η μεταβλητή είναι η *Περιοχή Επόμενης διεύθυνσης* του κόμβου, και θα αποθηκεύει την διεύθυνση του επόμενου κόμβου!

Σημείωση!

Θα μπορούσαμε να δηλώσουμε την μεταβλητή ConnectionAddress και ως τύπου δείκτη-void!.Όμως κάτι τέτοιο πραγματικά δεν συνίσταται καθόλου,καθώς οι δείκτες τύπου Void δεν χρησιμοποιούν την αριθμητική των δεικτών.Κάτι που πρέπει να αποφεύγεται καθώς μπορούμε με ένα λάθος βγούμε σε περιοχές μνήμης που δεν ανήκουν στον κόμβο μας ή ανήκουν σε κάπιο αλλο πρόγραμμα με αποτέλεσμα αλλοίωση δεδομένων και "κρέμασμα" της δομής δεδομένων μας !

Στο ρεζουμέ : Δημιουργώντας την γραμμική λίστα…

Για την δημιουργία, και την διαχείριση της γραμμικής μας λίστας , χρειάζονται συνολικά 6 συναρτήσεις , τις οποίες εμείς πρόκειται να δημιουργήσουμε ,δίνοντας έμφαση στις διάφορες μετατροπές τούς, ανάλογα με τον τύπο της γραμμικής λίστας που θέλουμε να υλοποιήσουμε

- Συνάρτηση : Push()
- Σύνταξη : Void Push (int NewNodeData)

Η συνάρτηση αυτή, θα δημιουργεί έναν καινούριο κόμβο και αφού θα του “περνάει” τα δεδομένα του θα τον προσαρμόζει κατάλληλα στην δομή,συνδέοντας τον με τους υπόλοιπούς κόμβους αυτόματα.Η συνάρτηση αυτή θα λαμβάνει 1 μόνο παράμετρο τύπου Integer η οποία θα αντιπροσωπεύει τα δεδομένα του καινούριου κόμβου

Σημείωση!

Σε περίπτωση που δηλώναμε στην δομή οτι η *περιοχή δεδομένων του κόμβου* θα ήταν αλλου τύπου (πχ float,char,bool κτλπ) τότε η συνάρτηση Push() προφανώς θα λάμβανε παράμετρο τύπου **Ιδιου** με τον τύπο του εκάστοτε κόμβου

Δημιουργώντας την Push()

Η διαδικασία εισαγωγής κόμβου σε μια δομή δεδομένων τύπου γραμμικής λίστας χωρίζεται σε δύο υποκατηγορίες,ανάλογα με την περίπτωση

1. Εισαγωγή κόμβου σε κενή λίστα (Ουσιαστικά *δημιουργία* της δομής)
2. Εισαγωγή κόμβου σε προυπάρχουσα λίστα

Εισαγωγή Κόμβου σε κενή λίστα

Η διαδικασία αυτή είναι εξαιρετικά απλή , καθώς μόλις τοποθετούμε τον κόμβο στην λίστα,τον δηλώνουμε ως κόμβο κεφαλής,αποθηκεύοντας την διεύθυνση του στην HeadNode.

```
1  #include <stdio.h> //Εισαγωγή του Standard Input/Output Header File της C
2  struct NewNode//Δήλωση δομής
3  {
4      int data; //Περιοχή Δεδομένων του κόμβου
5      struct NewNode *ConnectionAddress; //Περιοχή επόμενης διεύθυνσης
6
7
8  };//Τέλος Δήλωσης της δομής
9  struct NewNode *HeadNode = NULL;//Κομβος κεφαλής (Προσβασή στην δομή)
10
```

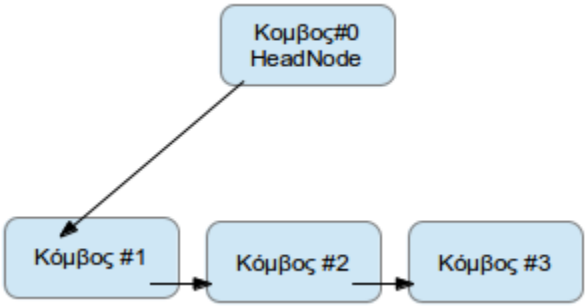
Τι είναι η HeadNode?

Πριν ξεκινήσουμε την Push(), **δηλώνουμε μια μεταβλητή-δείκτη τύπου NewNode με όνομα HeadNode**(Γραμμή 9) .Αυτός είναι ο δείκτης προς τον κόμβο κεφαλής.Αρχικά τον δηλώνουμε με τιμή

NULL και πάντοτε εκτός κάπιας συνάρτησης για να είναι ορατός απο ολο το πρόγραμμα (Καθολική εμβέλεια).Απο την HeadNode θα έχουμε πρόσβαση σε όλη την λίστα μας,καθώς θα είναι ο μοναδικός κόμβος που θα γνωρίζουμε πάντοτε την διεύθυνση του!.Η HeadNode είναι με άλλα λόγια το link στην μνήμη για να μπούμε στην λίστα μας!
Ανάλογα με την περίπτωση,διακρίνουμε δυο καταστάσεις του HeadNode

- Η HeadNode να ισούται με το NULL
  Αφού ο κόμβος κεφαλής είναι κενός , τότε δεν υπάρχει λίστα , έτσι γνωρίζουμε οτι αν ο κόμβος κεφαλής δεν περιέχει τίποτε τότε η λίστα μας είναι κενή!
- Η HeadNode να είναι διάφορη του NULL
  Πράγμα που σημαίνει προφανώς οτι η λίστα δεν είναι κενή !

Στην συνάρτηση Push() θα είναι καθοριστικό να ελένχουμε την HeadNode , έτσι θα γνωρίζουμε σε πια απο τις δύο περιπτώσεις βρισκόμαστε



Εναρκτήρια διεύθυνση της λίστας μας.

Όπως έχουμε προαναφέρει, κάθε κόμβος παρέχει σύνδεση με τον επόμενο του.Αυτό το αποτέλεσμα μας οδηγεί στο συμπέρασμα οτι, αν βρισκόμαστε σε κάπιον κόμβο,είναι αδύνατον να <<χαθούμε>> στην μνήμη.Έτσι για να μπούμε στην λίστα μας , χρειάζεται να γνωρίζουμε την διεύθυνση κάπιου κόμβου. Για τον σκόπο αυτό, υπάρχει ο δείκτης HeadNode, καθώς αυτή θα μας γνωστοποιεί την αρχική διεύθυνση της λίστας μας ,για να μπορούμε να την επεξεργαστούμε,όταν αυτό χρειαστεί. Όπως θα παρατηρήσετε , αρκετές συναρτήσεις λαμβάνουν ως παράμετρο την Εναρκτήρια διεύθυνση της λίστας μας

Πριν ξεκινήσουμε,ας αναλύσουμε τι πρόκειται να κάνουμε σε βήματα

1. Δεσμεύουμε δυναμικά μνήμη όσων bytes χρειαζόμαστε για να αποθηκεύσουμε έναν κόμβο, και αποθηκεύουμε εναν νέο κόμβο εκεί.
2. Ελένχουμε αν ο κόμβος κεφαλής είναι NULL
   - Αν είναι κενός τότε(η λίστα μας είναι κενή) δηλώνουμε τον καινούριο κόμβο ώς κόμβο κεφαλής και στην περιόχη επόμενης διεύθυνσης εισάγουμε την τιμή NULL (αφού ο πρώτος κόμβος θα είναι πάντοτε κόμβος ουράς (Tail Node)

     **Σημείωση! -> Όσο αναφορά λίστες χωρίς κανένα είδος κυκλικής σύνδεσης , η περιοχή επόμενης διεύθυνσης του κόμβου ουράς είναι πάντοτε NULL,καθώς έπειτα απο τον κόμβο ουράς δεν υπάρχει τίποτε άλλο!

   - Αν δεν είναι κενός,τότε ακολουθούμε την διαδικασία εισαγωγής κόμβου σε προυπάρχουσα δομή (που θα την εξηγήσουμε στην συνέχεια)

Ας δούμε τον (ημιτελη) κώδικα της Push() που υλοποιεί εισαγωγή κόμβου σε απλή γραμμική λίστα !

```
10
11  void Push(int NewNodeData)//Δήλωση της Push ως συνάρτησης τύπου int με παράμετρο NewNodeData
12                                                                  //επίσης τύπου int
13                          //Η παράμετρος θα περιέχει τα δεδομένα του νέου κόμβου
14  {
15      struct NewNode *NodeToEnter = (struct NewNode *)malloc(sizeof(struct NewNode));
16      //Δήλωση του Νεου Κόμβου(NodeToEnter) και δυναμική δέσμευση μνήμης
17      //ίση με το μέγεθος της δομής NewNode
18
19
20      NodeToEnter->data=NewNodeData; // Εισαγωγή των δεδομένων στον Νεο κόμβο κόμβο
21
22      if (HeadNode==NULL) // Αν η δομή μας είναι κενή
23      {
24          HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
25          HeadNode->ConnectionAddress=NULL;
26          return ;                // Και μήν κάνεις τίποτε αλλο
27      }
```

Εξηγώντας Γραμμή-Γραμμή τι κάναμε....

Γραμμή 11 : Δηλώνουμε την συνάρτηση Push() η οποία δεν θα επιστέφει δεδομένα(void),ως παράμετρο δηλώνουμε την μεταβλητή τύπου ακεραίων (int) NewNodeData η οποία θα αποθηκεύει τα δεδομένα που θέλουμε να εισάγουμε στον νέο μας κόμβο
Γραμμή 15 : Δημιουργούμε έναν νέο κόμβο και αποθηκεύουμε εκεί την εναρκτήρια διεύθυνση της δυναμικά δεσμευμένης μνήμης μεγέθους,όσο ακριβώς η δομη NewNode .Με λίγα λόγια δεσμεύσαμε όση μνήμη χρειαζόμαστε για να χωρέσει ένας κόμβος και αποθηκεύσαμε τον κόμβο εκεί!
Γραμμή 20 : Περνάμε τα δεδομένα int του κόμβου στην περιοχή δεδομένων του κόμβου με την χρήση του τελεστή παύλα-βέλος(->) ο οπόιος παρέχει πρόσβαση (αντί του τελεστή τελεία (.)) σε δομές απο μεταβλητές τύπου-δείκτη(πολύ θεωρία έπεσε ε? ;))
Γραμμή 22: Ελένχουμε αν η δομή δεδομένων μάς είναι κενή,εάν είναι τότε....
Γραμμή 24 : ...Απλά περνάμε τον κόμβο στην HeadNode,έτσι να τον δηλώσουμε ως τον πρώτο κόμβο!
Γραμμή 25 : Δηλώνουμε οτι η περιοχή της επόμενης διεύθυνσης του νεού κόμβου θα είναι NULL (καθώς ο πρώτος κόμβος θα είναι πάντοτε και κόμβος ουράς (Tail Node)

Εισαγωγή Κόμβου σε προυπάρχουσα λίστα

Μέχρι στιγμής έχουμε δημιουργήσει τον κώδικα της Push() η οποία υλοποιεί την εισαγωγή ενός κόμβου σε κενή λίστα,ηρθε λοιπόν η στιγμή να ολοκληρώσουμε αυτή την συνάρτηση τελειοποιώντας την , έτσι ώστε να εισάγει αυτόματα κόμβους και σε προυπάρχουσα λίστα

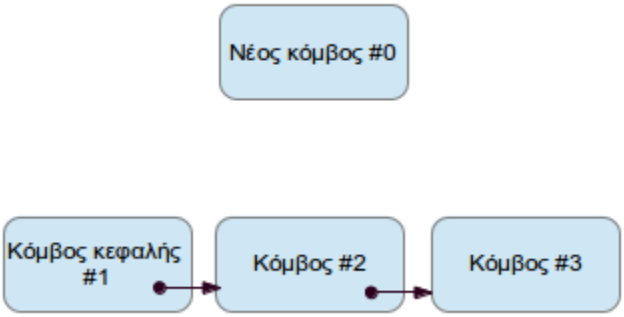
Αλγόριθμος εισαγωγής κόμβου σε προυπάρχουσα λίστα

Ο αλγόριθμος αυτός,είναι εξαιρετικά απλός και χωρίζεται σε 2 απλούστατα βήματα που έχουν ως εξής:

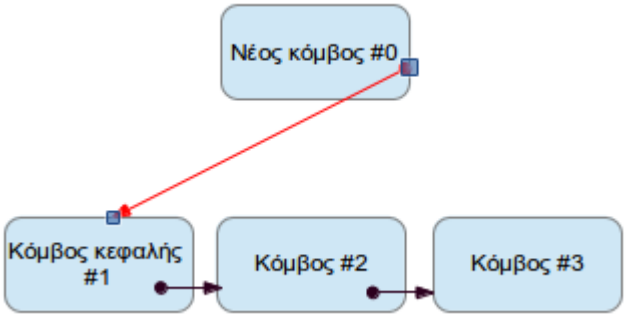
1. Δηλώνουμε την επόμενη διεύθυνση του νεόυ μας κόμβου τον υπάρχων κόμβο κεφαλής
2. Δηλώνουμε τον νεό μας κόμβο ως κόμβο κεφαλής

Ας το δούμε σχηματικά τι επρόκειτο να κάνουμε...

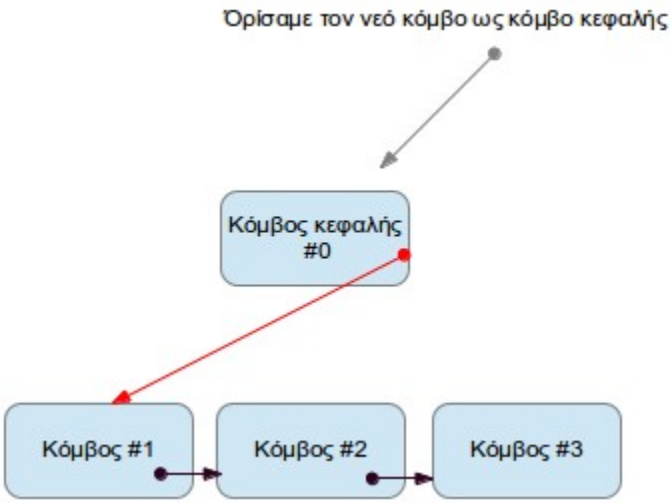
1)Δημιουργία νεόυ κόμβου .....



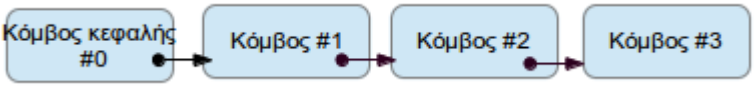
Αντιστοίχηση της περιοχής **επόμενης διεύθυνσης** του νέου κόμβου με τον κόμβο κεφαλής(**HeadNode).**



Ορισμός του νεού κόμβου ως κόμβος κεφαλής.



Και το τελικό αποτέλεσμα.η λίστα μας με τον νεό της κόμβο αυτόματα εισακτέο απο την Push() στην μνήμη.



Ολοκληρώνοντας την Push()

Ας δούμε τώρα πιά το πώς υλοποιείται η Push() .. Αναλύοντας τον αλγόριθμο που προαναφέρθηκε ο κώδικας μας θα ήταν κάπως έτσι....

```
10
11  void Push(int NewNodeData)//Δήλωση της Push ως συνάρτησης τύπου int με παράμετρο NewNodeData
12                                                          //επίσης τύπου int
13                          //Η παράμετρος θα περιέχει τα δεδομένα του νέου κόμβου
14  {
15      struct NewNode *NodeToEnter = (struct NewNode *)malloc(sizeof(struct NewNode));
16      //Δήλωση του Νεου Κόμβου(NodeToEnter) και δυναμική δέσμευση μνήμης
17      //ίση με το μέγεθος της δομής NewNode
18
19
20      NodeToEnter->data=NewNodeData; // Εισαγωγή των δεδομένων στον Νεο κόμβο κόμβο
21
22      if (HeadNode==NULL) // Αν η δομή μας είναι κενή
23      {
24          HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
25          HeadNode->ConnectionAddress=NULL;
26          return ;                // Και μήν κάνεις τίποτε αλλο
27      }
28      else{
29          NodeToEnter->ConnectionAddress = HeadNode;
30          HeadNode = NodeToEnter;
31
32
33
34      }
35
36
37  }
```

Εξηγώντας Γραμμή-Γραμμή...

Έχοντας εξηγήσει τις γραμμές 11-26 πιο πάνω ας εξηγήσουμε γραμμή γραμμή πως καταφέραμε να εισάγουμε τον νεό μας κόμβο στην προυπάρχουσα λίστα μας!

Γραμμή 28:Μια δομή επιλόγης if η οποία εξετάζει μέσω της HeadNode αν υπάρχει ήδη κόμβος μέσα στην γραμμική λίστα μας.

Γραμμή 29:Αντιστοιχούμε στην επόμενη διεύθυνση του νεού κόμβου την διεύθυνση του υπάρχοντος κόμβου κεφαλής....

Γραμμή : 30: ....Αποθηκεύουμε στην HeadNode την εναρκτήρια διεύθυνση του νεού κόμβου!

Έτσι η Push με μόνο μια κλήση της είναι ικάνη να διαχωρίσει πιον αλγόριθμο θα χρησιμοποιήσει κάθε φορά για να διαχειριστεί κατάλληλα την γραμμική λίστα μας!, Η Push() είναι έτοιμη για δράση!

Η Push για διπλά συνδεδεμένη λίστα

Η συνάρτηση push , με μερικές αλλαγές μπορεί να λειτουργεί και με διπλά συνδεδεμένες γραμμικές λίστες.....

ΤΙ ακριβώς χρειαζόμαστε;

Είναι προφανές, ότι για να κάνουμε την push να λειτουργεί και με διπλά συνδεδεμένες γραμμικές λίστες , θα πρέπει να δημιουργήσουμε μια *περιοχή προηγούμενης διεύθυνσης* στην δομή NewNode η οποία θα αποθηκεύει την διεύθυνση του προηγούμενου κόμβου.Έπειτα απο αυτό πρέπει να τροποποιήσουμε την push κατάλληλα έτσι ώστε να μπορεί να συνδεεί διπλά τους κόμβους

Βήμα 1:

Η τροποποίηση στην δομή των κόμβων περιλαμβάνει την προσθήκη μιας ακόμη μεταβλητής στην NewNode η οποία θα δείχνει προς διευθύνσεις της δομής NewNode.Στο σημείο αυτό θα ήταν συνετό να κάνατε μια επανάληψη στην δομή των κόμβων NewNode

```
1  #include <stdio.h> //Εισαγωγή του Standard Input/Output Header File της C
2  struct NewNode//Δήλωση δομής
3  {
4      int data; //Περιοχή Δεδομένων του κόμβου
5      struct NewNode *ConnectionAddress; //Περιοχή επόμενης διεύθυνσης
6      struct NewNode *ConnectionAddressBack //περιοχή προηγούμενης διεύθυνσης
7
8
9  };//Τέλος Δήλωσης της δομής
```

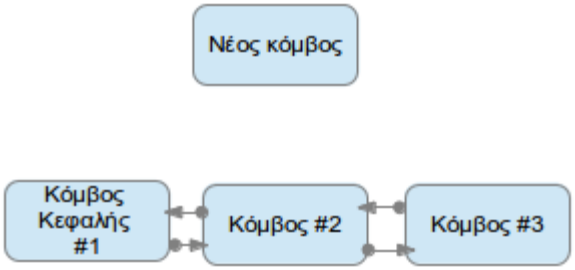
Βήμα 2:

Για να κάνουμε την Push να συνδεεί διπλά τους κόμβους ,θα πρέπει

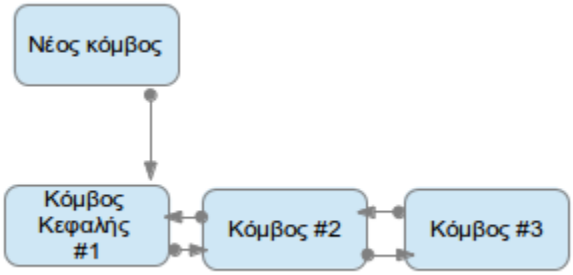
- Στην περίπτωση που η HeadNode είναι NULL (δηλαδή που η λίστα μας είναι απολύτως κενή) να δηλώνεται ως NULL,οχι μόνο η περιοχή επόμενης διεύθυνσης(Γραμμή 47) αλλα και η περιοχή προηγούμενης διεύθυνσης(Γραμμή 48), λόγω ότι ο πρώτος κόμβος δεν έχει ούτε επόμενο , ούτε προηγούμενο κόμβο
  Ας δούμε την τροποποίηση της μίας γραμμής στην push(Γραμμή 48)

```
40 void push_double_linked_list(int NewNodeData)
41 {
42     struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
43     NodeToEnter->data=NewNodeData;
44     if (HeadNode==NULL) // Αν η δομή μας είναι κενή
45     {
46         HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
47         HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
48         HeadNode->ConnectionAddressBack=NULL; //προηγούμενη διεύθυνση κενή
49         return ;                // Και μήν κάνεις τίποτε αλλο
50     }
51     else{
52         NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
53         HeadNode = NodeToEnter;                    //Δήλωση νέου κόμβου ως κόμβου HeadNode
54
55
56
57     }
58
59 }
60
```

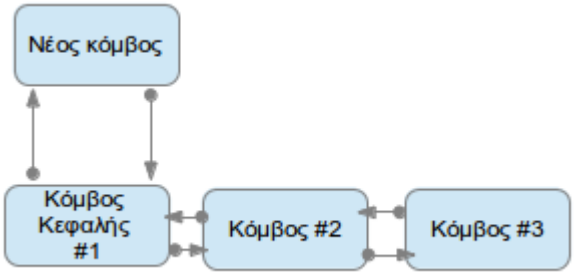
- Στην περίπτωση που η HeadNode δεν είναι NULL(Δηλαδή που η λίστα μας δεν είναι κενή) θα πρέπει σε κάθε φορά που <<περνάμε>> έναν κόμβο στην δομή , να ενημερώνουμε τον επόμενο κόμβο οτι ο προηγούμενος του είναι ο νεοεισαχθέντας κόμβος! Και επείδη είμαι σίγουρός οτι δεν το καταλάβατε με την πρώτη, ας το δούμε καλύτερα σχηματικά.
  Έστω η παρακάτω δομή



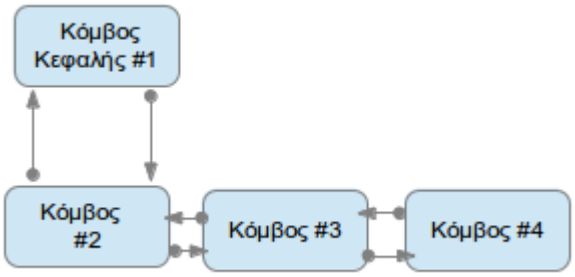
Πρώτον , ενώνουμε τον νεό κόμβο με τον υπάρχων κόμβο κεφαλής μέσω της *περιοχής επόμενης διεύθυνσης* όπως ακριβώς κάνουμε και με την απλή συνδεδεμένη λίστα



Έπειτα Γνωστοποιούμε στον κόμβο κεφαλής *μέσω της περιοχής προηγούμενης διεύθυνσης* οτι ο προηγούμενος κόμβος του είναι ο νέος κόμβος



Τέλος , ορίζουμε τον νεό κόμβο ώς κόμβο κεφαλής



Taik is cheap! Show me the code (Linus Torvalds 25-08-00)

Ας δούμε λοιπόν την τροποποίηση της Push,η οποία υλοποιεί την διπλά συνδεδεμένη λίστα,Δώστε ιδιέτερη προσοχή στην ουσιαστική τροποποίηση της γραμμής 53,καθώς εκεί ενημερώνουμε τον προηγούμενο κόμβο για τον σχέση του με τον νεό.

```
40  void push_double_linked_list(int NewNodeData)
41  {
42      struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
43      NodeToEnter->data=NewNodeData;
44      if (HeadNode==NULL) // Αν η δομή μας είναι κενή
45      {
46          HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
47          HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
48          HeadNode->ConnectionAddressBack=NULL; //προηγούμενη διεύθυνση κενή
49          return ;                // Και μήν κάνεις τίποτε αλλο
50      }
51      else{
52          NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
53          HeadNode->ConnectionAddressBack=NodeToEnter; // Προηγούμενος του HeadNode ο νεος
54          HeadNode = NodeToEnter;                     //Δήλωση νέου κόμβου ως κόμβου HeadNode
55
56          |
57
58      }
59
60  }
```

Η Push για κυκλικά συνδεδεμένη λίστα

Η τροποποίηση της Push για κυκλικά συνδεδεμένη λίστα είναι μια αρκετά εύκολη υπόθεση , η οποία περιλαμβάνει τα εξής βήματα

- Δημιουργούμε μια μεταβλητή δείκτη προς την δομή NewNode . Η οποία θα ονομάζεται TailNode και θα δείχνει πάντοτε τον κόμβο ουράς

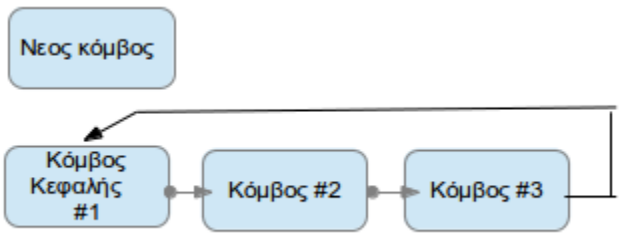
Αυτό γίνεται με μονάχα μια γραμμή κώδικα , η οποία δηλώνει μια τέτοιου τύπου μεταβλητή (Γραμμή 11)

```c
#include <stdio.h> //Εισαγωγή του Standard Input/Output Header File της C
struct NewNode//Δήλωση δομής
{
    int data; //Περιοχή Δεδομένων του κόμβου
    struct NewNode *ConnectionAddress; //Περιοχή επόμενης διεύθυνσης
    struct NewNode *ConnectionAddressBack; //περιοχή προηγούμενης διεύθυνσης


};//Τέλος Δήλωσης της δομής
struct NewNode *HeadNode = NULL;//Κομβος κεφαλής (Προσβασή στην δομή)
struct NewNode *TailNode = NULL;//Κόμβος ουράς
```

- Τροποποιούμε κατάλληλα την push,έτσι ώστε πάντοτε η περιοχή επόμενης διεύθυνσης του κόμβου ουράς, να δείχνει τον κόμβο κεφαλής!
    - Στην περίπτωση που η HeadNode είναι NULL (δηλαδή που η λίστα μας είναι απολύτως κενή) , τότε ο νεοεισαχθέντας κόμβος δηλώνεται ως κόμβος ουράς και κόμβος κεφαλής ταυτόχρονα(Γραμμή 70)

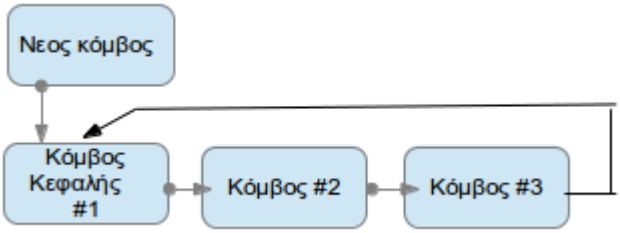
```c
void push_circular_linked_list(int NewNodeData)
{
    struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
    NodeToEnter->data=NewNodeData;
    if (HeadNode==NULL) // Αν η δομή μας είναι κενή
    {
        HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
        TailNode = NodeToEnter; //και ως κόμβο ουράς
        HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
        return ;                // Και μήν κάνεις τίποτε αλλο
    }
    else{
        NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
        HeadNode->ConnectionAddressBack=NodeToEnter; // Προηγούμενος του HeadNode ο νεος
        HeadNode = NodeToEnter;                     //Δήλωση νέου κόμβου ως κόμβου HeadNode


    }

}
```

    - Στην περίπτωση που η HeadNode δεν είναι NULL(Δηλαδή που η λίστα μας δεν είναι κενή) θα πρέπει κάθε φορά που εισάγουμε νεό κόμβο , να ενημερώνουμε την *περιοχή επόμενης διεύθυνσης* του *κόμβου ουράς* οτι ο επόμενος κόμβος της είναι ο νεοεισαχθέντας κόμβος κεφαλής HeadNode
    Ας το δούμε σχηματικά...
    Έστω η παρακάτω διπλά συνδεδεμένη λίστα

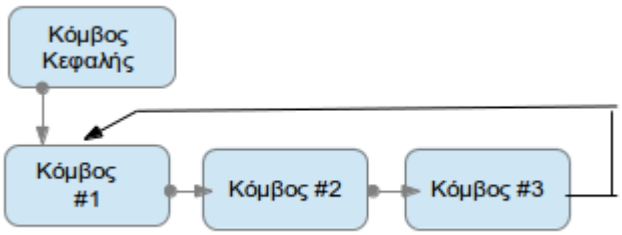


Βήμα πρώτο : Δηλώνουμε οτι ο επόμενος κόμβος του νεοεισαχθέντα κόμβου ειναι ο

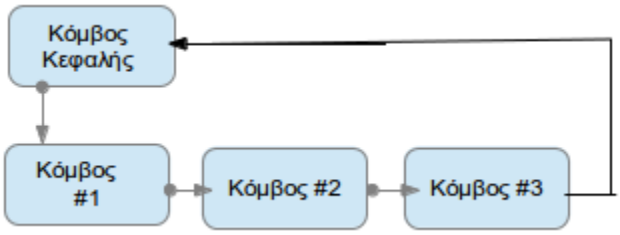
κόμβος κεφαλής



Βήμα δεύτερο : Δηλώνουμε τον νεό κόμβο ως *κόμβο κεφαλής*



Βήμα τρίτο : Δηλώνουμε στην *περιοχή επόμενης διεύθυνσης* του *κόμβου ουράς* οτι ο επόμενος κόμβος του είναι ο κόμβος κεφαλής



Έτσι λοιπόν ας κάνουμε την απαραίτητη αλλαγη στην Push ,στην γραμμή 77

```c
void push_circular_linked_list(int NewNodeData)
{
    struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
    NodeToEnter->data=NewNodeData;
    if (HeadNode==NULL) // Αν η δομή μας είναι κενή
    {
        HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
        TailNode = NodeToEnter; //και ως κόμβο ουράς
        HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
        return ;                // Και μήν κάνεις τίποτε αλλο
    }
    else{
        NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
        HeadNode = NodeToEnter;                     //Δήλωση νέου κόμβου ως κόμβου HeadNode
        TailNode->ConnectionAddress=HeadNode;



    }

}
```

Η Push για κυκλικα-διπλά συνδεδεμένη λίστα

Γνωρίζοντας την υλοποίηση της κυκλικής συνδεδεμένης λίστας και της διπλής συνδεδεμένης λίστας , είναι αρκετά εύκολο να συνδιάσουμε αυτα τα δύο για να δημιουργήσουμε την κυκλικά-διπλά συνδεδεμένη λίστα , ως εξής

Για ευνόητους λόγους , δεν αναλυώ διεξόδικά τον κώδικα , καθώς αυτό έχει γίνει ήδη παραπάνω

```c
void push_circular_double_linked_list(int NewNodeData)
{
    struct NewNode *NodeToEnter = (struct NewNode *)malloc(sizeof(struct NewNode));
    NodeToEnter->data = NewNodeData;
    if(HeadNode == NULL)
    {

        HeadNode = NodeToEnter;
        TailNode = NodeToEnter;
        NodeToEnter->ConnectionAddress=NULL;
        NodeToEnter->ConnectionAddressBack=NULL;
        return;

    }
    else
    {
        NodeToEnter->ConnectionAddress=HeadNode;
        HeadNode->ConnectionAddressBack = NodeToEnter;
        HeadNode=NodeToEnter;
        TailNode->ConnectionAddress=HeadNode;

    }
}

```

Τεστάροντας την δομή δεδομένων μας !

Μπορούμε με έναν εύκολο και κομψό τρόπο να τεστάρουμε την δομή δεδομένων μας για την πλήρη λειτουργικότητα της

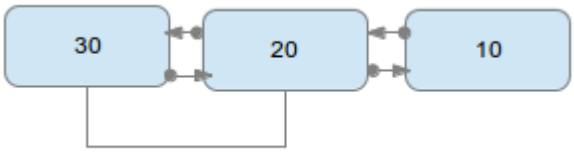
Σε διπλά συνδεδεμένη λίστα ας δοκιμάσουμε τις παρακάτω εντολές,οι οποίες βρίσκονται στην συνάρτηση main ()

```
147  int main()
148  {
149      push_double_linked_list(10);
150      push_double_linked_list(20);
151      push_double_linked_list(30);
152      printf("%d",HeadNode->ConnectionAddress->ConnectionAddress->ConnectionAddressBack->data);
153  }
```

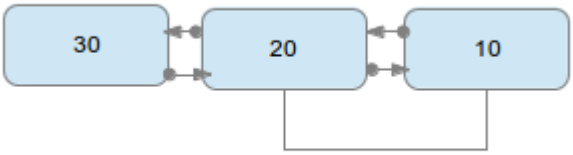
Παρατηρήστε το πόσο όμορφα εναλασόμαστε μέσα στους κόμβους,μέσω των περιοχών επόμενων διευθύνσεων! Το παρακάτω σύνολο εντολών αφού δημιουργεί μια δομή ως εξής

Εναλάσεται διαδοχικά απο τον κόμβο κεφαλής στον επόμενο του (δηλαδή αυτόν με τα δεδομένα 20) και στον πιο επόμενο απο αυτόν (με δεδομένα 10).Τέλος μέσω της περιοχής *προηγούμενης διεύθυνσης* ξαναγυρνάει στον προηγούμενο του κόμβο (με δεδομένα 20) . Έτσι συμβαίνει το εξής

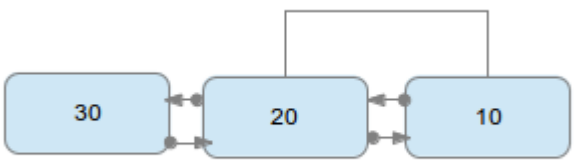
Απο το HeadNode (30) στον δεύτερο κόμβο (20)



Απο τον δευτερό κόμβο (20) στον τρίτο κόμβο (10)



Και απο τον τρίτο κόμβο(10) στον δεύτερο κόμβο (20)



Έτσι το τελικό αποτέλεσμα μας είναι ...

```
20
atticadreamer@atticadreamer-desktop:
```

Φίνα!Η push παίρνει 10 στα 10! αυτό ακριβώς που περιμέναμε!

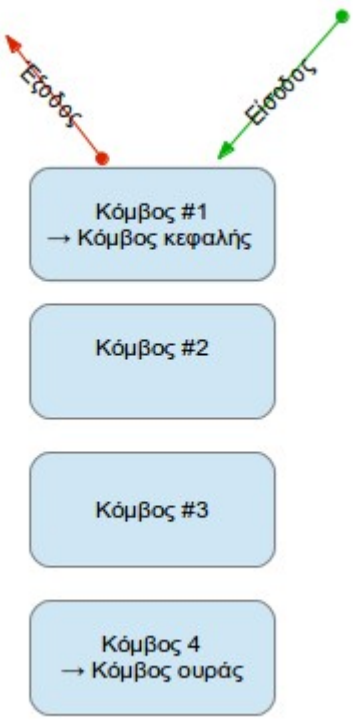
- Συνάρτηση : Pop()
- Σύνταξη : Void pop()

Η συνάρτηση αυτή δεν λάμβάνει παραμέτρους ,δουλεία της είναι , ύστερα απο την κλήση της ,να κάνει το ακριβώς ανάποδο απο οτι η Push! Θα διαγράφει κόμβους απο την λίστα μας .

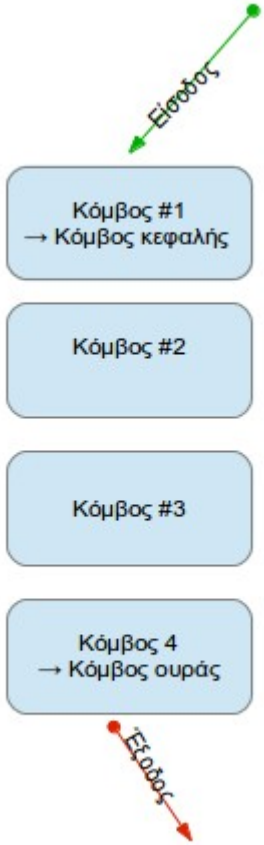
Λίγη (Ακόμη)Θεωρία

Όσο αναφορά την συνάρτηση pop () η συνάρτηση η ίδια έχει δύο <<εκδόσεις>> ανάλογα με την αρχιτεκτονική που θέλουμε να ακολουθήσουμε
Ας δούμε τις δύο αυτές *αρχιτεκτονικές* της συνάρτησης Pop()

- Αρχιτεκτονική *Στοίβας(LIFO)*
  Η Αρχιτεκτονική αυτή*(LIFO → Last In First Out)* λειτουργεί ως εξής : Σε κάθε κλήση της pop() αφαίρεσε τον τελευταίο κόμβο που προσαρτήθηκε στην λίστα,και όρισε τον κόμβο κεφαλής τον επόμενο απο τον κόμβο που αφαιρέθηκε.η συνάρτηση αυτή είναι συμβατή με όλους τους τύπους γραμμικών λιστών



- Αρχιτεκτονική *Ουράς(FIFO)*
  Η Αρχιτεκτονική αυτή*(FIFO → First In First Out)* λειτουργεί ως εξής : Σε κάθε κλήση της pop() αφαίρεσε τον πρώτο κόμβο που προσαρτήθηκε στην λίστα,και όρισε τον κόμβο ουράς τον προηγούμενο κόμβο απο τον κόμβο που αφαιρέθηκε.



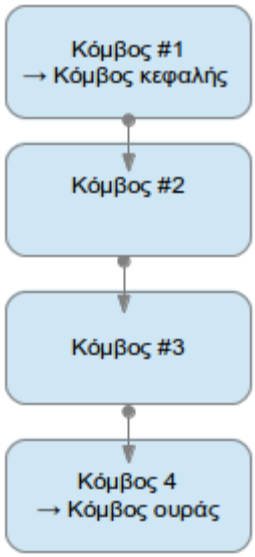
Δημιουργώντας την συνάρτηση pop()

Αρχιτεκτονική στοίβας

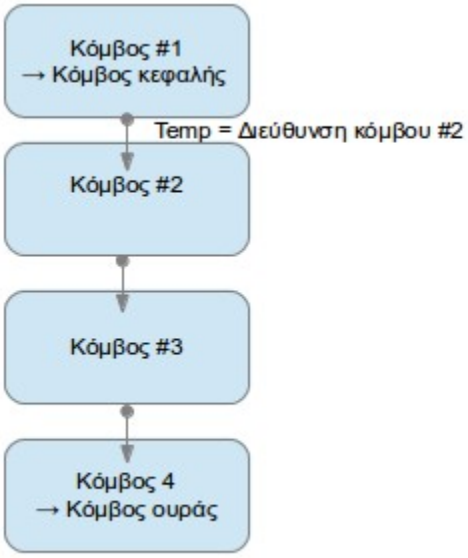
Η αρχιτεκτονική στοίβας προβλέπει μια αρκετά εύκολη pop() , η οποία αφαίρει τον πρώτο κόμβο ακολουθώντας τα εξής βήματα

1. *Αποθήκευσε την διεύθυνση του επόμενου κόμβου απο τον κόμβο κεφαλής σε μια μεταβλητή*
2. *Διέγραψε τον κόμβο κεφαλής*
3. *Όρισε ως κόμβο κεφαλής την διεύθυνση που αποθήκευσες στο βήμα 1*

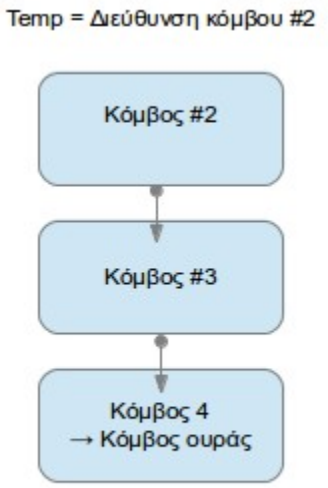
Ας δούμε σχηματικά τι εννοούμε
Έστω η δομή



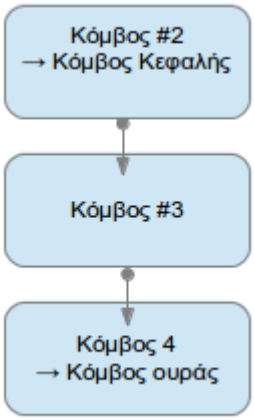
Βήμα 1) Αποθήκευση του επόμενου κόμβου απο τον κόμβο κεφαλής



Βήμα 2)Διαγραφή του κόμβου κεφαλής



Βήμα 3) Όρισμος της αποθηκευμένης διεύθυνσης ως κόμβος κεφαλής



Ας το δούμε στην πράξη...

Ας δούμε τον κώδικα που υλοποιεί την pop() αρχιτεκτονικής στοίβας και ας εξηγήσουμε τι κάναμε γραμμή-γραμμή

```
210  void pop_lifo()
211  {
212      struct NewNode *TheNextHeadNode = HeadNode->ConnectionAddress;
213      free(HeadNode);
214      HeadNode = TheNextHeadNode;
215      //TailNode->ConnectionAddress = HeadNode;
216  }
```

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 210 : Δήλωση της Pop_lifo ως μιας συνάρτησης τύπου void
Γραμμή 212:Αποθήκευση της διεύθυνσης του επόμενου κόμβου απο τον κόμβο κεφαλής (Βήμα 1)
Γραμμή 213 : Διαγραφή του κόμβου κεφαλής με την χρήση της συνάρτησης free()(Βήμα 2)
Γραμμή 214 : Επαναπροσδιορισμός του κόμβου κεφαλής ως ο επόμενος κόμβος απο αυτόν που διαγράφτηκε , όπως προαναφέραμε στο βήμα 3
Γραμμή 215: Σε περίπτωση κυκλικα συνδεδεμένης λίστας , πρέπει να ενημερώσουμε τον κόμβο ουράς ,για την νεα διεύθυνση του κόμβου κεφαλής.

Δημιουργώντας την συνάρτηση pop() (vol. 2)

Αρχιτεκτονική ουράς

Η αρχιτεκτονική αυτή, είναι συμβατή μόνο με τις διπλα-συνδεδεμένες λίστες .για τον ευνόητο λόγο οτι μόνο έτσι μπορούμε να αποθηκεύσουμε την διεύθυνση του προηγούμενου κόμβου , απο τον κόμβο ουράς.
*Εδώ να σημειώσουμε,ότι χρειαζόμαστε έναν δείκτη προς τον κόμβο ουράς(TailNode),έτσι ώστε να εφαρμόσουμε αυτού του είδους την αρχιτεκτονική της pop(),άρα χρειαζόμαστε μια μετατροπή οποιασδήποτε push για να μπορούμε να έχουμε δείκτη προς τον κόμβο ουράς.

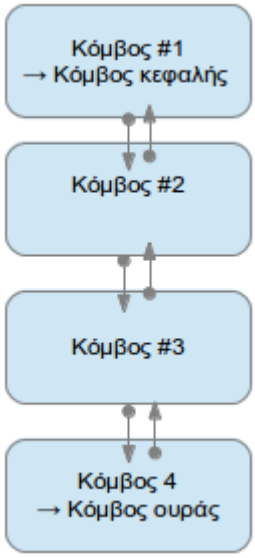
Για να υλοποιήσουμε λοιπόν αυτή την αρχιτεκτονική, χρειαζόμαστε συνολικά 3 βήματα

1. Αποθήκευση του προηγούμενου κόμβου , απο τον κόμβο ουράς , σε μία μεταβλητή δείκτη
2. Διαγραφή του κόμβου ουράς

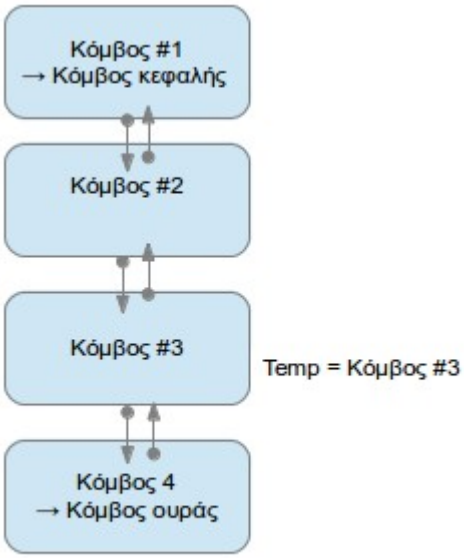
3. ορισμός ως κόμβος ουράς τον κόμβο που αποθηκεύσαμε στο βήμα 1

Ας το δούμε σχηματικα....

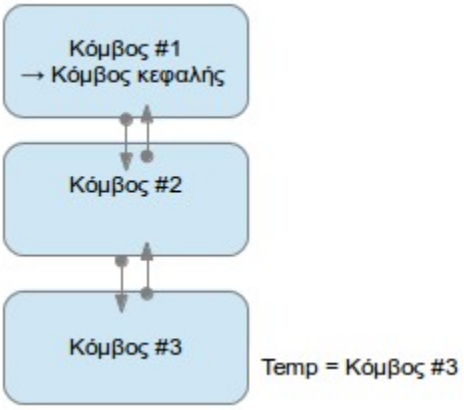
Έστω η παρακάτω δομή



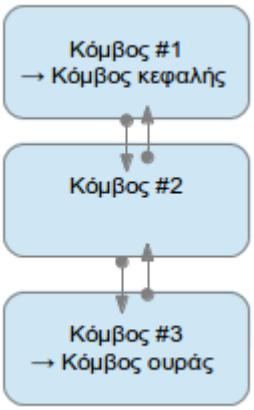
Βήμα 1) Αποθηκεύουμε την διεύθυνση του προηγούμενου κόμβου , απο τον κόμβο ουράς.



Βήμα 2) Διαγράφουμε τον κόμβο ουράς



Βήμα 3) Ορίζουμε τον κόμβο ουράς ως τον κόμβο που αποθηκεύσαμε στο βήμα 1



Lets write some code ;)

Πρίν εφαρμόσουμε την αρχιτεκτονική αυτή , είναι απαραίτητη μια μετατροπή την συνάρτηση push() για διπλα συνδεδεμένη λίστα , έτσι ώστε να υπάρχει και η μεταβλητή-δείκτης TailNode προς τον κόμβο ουράς

Έτσι έχουμε μια μικρή μετατροπή(γραμμή 52),έτσι ώστε να αποθηκεύουμε την διευθύνση του κόμβου ουράς !

```
43  void push_double_linked_list(int NewNodeData)
44  {
45      struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
46      NodeToEnter->data=NewNodeData;
47      if (HeadNode==NULL) // Αν η δομή μας είναι κενή
48      {
49          HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
50          HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
51          HeadNode->ConnectionAddressBack=NULL; //προηγούμενη διεύθυνση κενή
52          TailNode = NodeToEnter;
53          return ;                // Και μήν κάνεις τίποτε αλλο
54      }
55      else{
56          NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
57          HeadNode->ConnectionAddressBack=NodeToEnter; // Προηγούμενος του HeadNode ο νεος
58          HeadNode = NodeToEnter;                     //Δήλωση νέου κόμβου ως κόμβου HeadNode
59
60
61
62      }
63
64  }
```

Ας δούμε τον κώδικα της pop() αρχιτεκτονικής ουράς....

```
220  void pop_fifo()
221  {
222      struct NewNode *TheNextTailNode = TailNode->ConnectionAddressBack;
223      free(TailNode);
224      TailNode = TheNextTailNode;
225      //TailNode->ConnectionAddress = HeadNode;
226      //HeadNode->ConnectionAddressBack = TailNode;
227  }
```

Εξηγώντας γραμμή-γραμμή

Γραμμή 222 . Αποθήκευση του προηγούμενου κόμβου , απο τον κόμβο ουράς , στην μεταβλητη TheNextTailNode

Γραμμή 223 : Διαγραφή του κόμβου ουράς

Γραμμή 224 : Ορισμός του κόμβου ουράς , ώς του προηγούμενου κόμβου απο αυτόν που διαγράφτηκε (Βήμα 2)

Γραμμή 225:Στην περίπτωση κυκλικά συνδεδεμένης λίστας , οι κόμβοι ουράς και κεφαλής ενημερώνωνται για τους νεούς επόμενους και προηγούμενους κόμβους τους αντίστοιχα . Ο κώδικας αυτός χρησιμοποιείται μόνο , σε περίπτωση που εφαρμόσουμε την συνάρτηση αυτή , σε κυκλικά συνδεδεμένη λίστα , έτσι ώστε και μετα απο μια διαγραφή του κόμβου

Ας τεστάρουμε τις pop() που έχουμε δημιουργήσει !

Έστω σε μια διπλά συνδεδεμένη λίστα καλούμε την pop() αρχιτεκτονικής στοίβας ως εξής

```
231  int main()
232  {
233      push_double_linked_list(10);
234      push_double_linked_list(20);
235      push_double_linked_list(30);
236      pop_lifo();
237      printf("%d \n",HeadNode->data);
238
239
240  }
```

Όπως είναι προφανές,η συνάρτηση pop_lifo() θα αφαιρέσει τον κόμβο κεφαλής (τον κόμβο με δεδομένα τον αριθμό 30) και στην θέση του θα βάλει τον επόμενο του (τον κόμβο με δεδομένα τον αριθμό 20) . Έτσι ζωτώντας να εκτυπωθεί ο κόμβος κεφαλής (Γραμμή 237) θα μας εκτυπωθεί ο αριθμος 20
Πράγμα που γίνεται φυσικά

```
20
atticadreamer@atticadreamer-desktop:~/OnProgressProjects$
```

Άν τώρα καλέσουμε την pop_fifo() τότε προφανώς θα λείπει ο κόμβος ουράς , έτσι αντί για τον κόμβο με δεδομένα τον αριθμό 10 , κόμβος ουράς θα είναι ο κόμβος , με αριθμό 20! .και ούτω κάθε εξής !

- # Συνάρτηση Display()
- Σύνταξη : void Display(struct NewNode *HeadAddress)

Η συνάρτηση αυτή λαμβάνει μόνο μια παράμετρο,η οποία είναι η εναρκτήρια διεύθυνση της λίστας μας ,και επιστρέφει ως αποτέλεσμα την εκτύπωση όλων των δεδομένων όλων των κόμβων διαβάζοντας τους σειριακά

Δημιουργώντας την Display()

Η Display() είναι μια αρκετά ευκολη υπόθεση.καθώς απαρτίζεται απο μόλις 9 γραμμές κώδικα.Η λόγική είναι η εξής : Ξεκινώντας απο τον κόμβο κεφαλής,εκτυπώνουμε το περιεχόμενο του , και προχωράμε στον επόμενο,μέσω της περιοχής επόμενης διεύθυνσης .Αυτό επαναλαμβάνεται μέχρι στην περιοχή επόμενης διεύθυνσης συναντήσουμε την τιμή NULL (Δηλαδή όταν έχουμε φτάσει στον κόμβο ουράς) .
Ας δούμε τον κώδικα που υλοποιεί την Display() και έπειτα ας εξηγήσουμε γραμμη-γραμμή τι κάναμε

```
110  void Display(struct NewNode *HeadAddress)
111  {
112      struct NewNode *p;
113      p = HeadAddress;
114      while (p->ConnectionAddress != NULL){
115          printf("%d\n",p->data);
116          p=p->ConnectionAddress;
117      }
118      printf("%d\n",p->data);
119  }
120
```

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 110 : Δηλώνουμε την συνάρτηση ως void με παράμετρο την *εναρκτήρια διεθυνση* της λίστας μας!
Γραμμή 112: Δηλώνουμε την μεταβλητή-δείκτη η οποία θα χρησιμέυσει για να σαρώνει την δομή δεδομένων μας.Φανταστείτε οτι αυτή η μεταβλητή θα παίζει τον ρόλο του μετρητή οταν σαρώνουμε ένα απλό πίνακα.
Γραμμή 113:Η μεταβλητή – δείκτη λαμβάνει την αρχική τιμή HeadAddress η οποία είναι η αρχική διεύθυνση της δομής που θέλουμε να εκτυπώσουμε
Γραμμή 114: Επαναληπτική δομή while() η οποία θα τερματίσει μόνο όταν η επόμενη διεύθυνση του

κόμβου που βρίσκομαστε είναι NULL (Όταν δηλαδή έχουμε φτάσει στον κόμβο ουράς)
Γραμμή 115:Εκτυπώνουμε τα δεδομένα του κόμβου που βρισκόμαστε
Γραμμή 116: Η p ισούται με την διεύθυνση του επόμενου κόμβου(Μέσω της μεταβλητής επόμενης διεύθυνσης) .ουσιαστικά προχωρούμε στον επόμενο κόμβο!
Γραμμη 118: ΕΔΩ είναι ενα εύλογο ερώτημα!Γιατί αλλη μία printf? Η απάντηση είναι απλή. Όταν βρισκόμαστε στον τελευταίο κόμβο.τότε η επόμενη διεύθυνση του είναι NULL! Έτσι σταματάμε την επανάληψη πρίν προλάβουμε να εκτυπώσουμε τα δεδομένα του τελευταίου κόμβου.Όπότε πάντοτε εκτός της επαναληπτικής διαδικάσίας while βάζουμε ενα printf(),για να εκτυπωθούνε τα δεδομένα του τελευταίου κόμβου

Αν όλα πάνε καλά...

και έχετε κάνει ολα αυτά τα οποία εξηγήσαμε , πίστά και χωρίς λάθη , τότε το αποτέλεσμα των παρακάτω εντολών....

```
147  int main()
148  {
149      Push(10);
150      Push(20);
151      Push(30);
152      Display(HeadNode);
153  }
```

Θα έχει έξοδο ως εξής .....

```
atticadreamer@atticadreamer-desktop:~$ ./a.out
30
20
10
atticadreamer@atticadreamer-desktop:~$
```

Όσο αναφορά τις μετατροπές ....

Η συνάρτηση Display() χρειάζεται μετατροπή μόνο στις κυκλικά συνδεδεμένες λίστες, και αυτό για τον απλούστατο λόγο ,οτι μια κυκλικά συνδεδεμένη λίστα ,δεν περιέχει NULL στον κόμβο ουράς,οπότε μια κλήση της Display(),θα μας έφερνε αντιμέτωπους μια μια ατέρμονη σάρωση και εκτύπωση της λίστας μας !

Μια προφανής λύση είναι η εξής

Δίνουμε στην δομή των κόμβων NewNode ακόμη ενα αριθμητικό πεδίο που θα έχει ως εξής

- Αν είναι 1 τότε ο κόμβος θα είναι κόμβος ουράς
- Αν είναι 0 τότε θα είναι κάπιος άλλος κόμβος!

Έτσι η Display θα ανιχνεύει αν βρίσκεται στον κόμβο ουράς , όχι απο την περιοχή *επόμενης διεύθυνσης* , αλλα απο το νεό αριθμητικό πεδίο που ορίσαμε !
Οπότε έχουμε διαδοχικά

1) Την δομή , όπου στην γραμμή 7 δηλώνεται το νέο πεδίο

```
1  #include <stdio.h> //Εισαγωγή του Standard Input/Output Header File της C
2  struct NewNode//Δήλωση δομής
3  {
4      int data; //Περιοχή Δεδομένων του κόμβου
5      struct NewNode *ConnectionAddress; //Περιοχή επόμενης διεύθυνσης
6      struct NewNode *ConnectionAddressBack; //περιοχή προηγούμενης διεύθυνσης
7      int isTail;
8
9
10 };//Τέλος Δήλωσης της δομής
```

2)Την τροποποιημένη συνάρτηση Push της διπλά συνδεδεμένης λίστας , έτσι ώστε να <<μαρκάρει>> τον κόμβο ουράς (Γραμμές 74 και 81)

```
65 void push_circular_linked_list_edited(int NewNodeData)
66 {
67     struct NewNode *NodeToEnter =(struct NewNode *)malloc(sizeof(struct NewNode));
68     NodeToEnter->data=NewNodeData;
69     if (HeadNode==NULL) // Αν η δομή μας είναι κενή
70     {
71         HeadNode = NodeToEnter; // Δήλωσε τον νέο κόμβο ώς κόμβο κεφαλής
72         TailNode = NodeToEnter; //και ως κόμβο ουράς
73         HeadNode->ConnectionAddress=NULL; //Επόμενη διεύθυνση κενή
74         NodeToEnter->isTail = 1; //δηλώσε οτι είναι κόμβος ουράς
75         return ;                // Και μήν κάνεις τίποτε αλλο
76     }
77     else{
78         NodeToEnter->ConnectionAddress = HeadNode; // επόμενος του Νεου κόμβου ο HeadNode
79         HeadNode = NodeToEnter;                    //Δήλωση νέου κόμβου ως κόμβου HeadNode
80         TailNode->ConnectionAddress=HeadNode;      //δημιουργία της κυκλικης σύνδεσης
81         NodeToEnter->isTail = 0; //δεν είναι κόμβος ουράς
82
83
84
85
86     }
87
88 }
89
```

Όπως είναι προφανές , ό πρώτος κόμβος που θα εισαχθεί στην λίστα μας , θα είναι πάντοτε ο ουριαίος κόμβος . Οπότε αυτός μαρκάρεται ως κόμβος ουράς με την τιμή 1 στο αριθμητικό πεδίο isTail

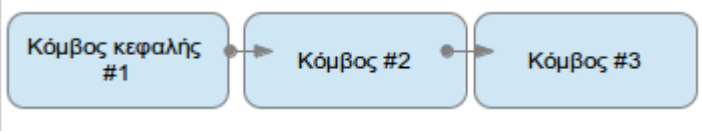
3) Την τροποποιημένη Display η οποία σταματάει , όχι εκεί που βρίσκει NULL ,αλλα εκεί που ο κόμβος έχει το πεδίο isTail να ισούται με 1(Γραμμή 150)

```
146  void Display_CL(struct NewNode *HeadAddress)
147  {
148      struct NewNode *p;
149      p = HeadAddress;
150      while (p->isTail!=1){
151          printf("%d\n",p->data);
152          p=p->ConnectionAddress;
153      }
154      printf("%d\n",p->data);
155  }
```

*Τα αρχικά CL στο όνομα της Display σημαίνουν Circular List

- # Συνάρτηση Index()
- Σύνταξη : int Index (struct NewNode *HeadAddress , int Position)

Η συνάρτηση αυτή λαμβάνει δύο παραμέτρους,η μία είναι η αρχική διεύθυνση του κόμβου κεφαλής , και η δεύτερη , ο αριθμός του κόμβου μέσα στην δομή προς εκτύπωση. Δουλεία της Index() είναι να σαρώνει την δομή και να επιστέφει τα δεδομένα ενός συγκεκριμένου κόμβου , που τον καλούμε απο την αυξουσα θέση του μέσα στην δομή.Παραδειγματικά
Έστω η παρακάτω λίστα



η κλήση της Index() με παραμέτρους την διεύθυνση του κόμβου κεφαλής , και τον αριθμό 0 . θα μας επιστρέψει τα δεδομένα του κόμβου κεφαλής , καθώς αυτός είναι ο πρώτος κόμβος της λίστας μας!

Δημιουργώντας την Index()

Η Index() είναι αρκετά παρομοιότυπη με την Display(). Αλλα διαφέρει στο γεγονός οτι εκτυπώνει τα δεδομένα ενός συγκέκριμένου κόμβου έναντι απο όλη την λίστα . Η Index() λειτουργεί ως εξής : <<ορίζουμε εναν μετρητή με την τιμή 0. Ξεκινώντας απο τον κόμβο κεφαλής,πρωχωράμε στον επόμενο κόμβο ,μέσω της περιοχής επόμενης διεύθυνσης του.Αυτό επαναλαμβάνεται μέχρι ο μετρήτης να είναι ίσος με την τιμη που μας έχει δωθεί ως παράμετρος (Position) .Τέλος εκτυπώνουμε την τιμή του κόμβου που σταματήσαμε >>
Ας δούμε τον κώδικα που υλοποιεί την Index() και έπειτα ας εξηγήσουμε γραμμη-γραμμή τι κάναμε

```
50  void Index(struct NewNode *HeadAddress , int position)
51  {
52      struct NewNode *p;
53      p = HeadAddress;
54      int i;
55      for(i=0;i<position;i++){
56          p=p->ConnectionAddress;
57      }
58      printf("%d\n",p->data);
59  }
```

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 52: Δηλώνουμε την μεταβλητή-δείκτη η οποία θα χρησιμέυσει για να σαρώνει την δομή δεδομένων μας.Φανταστείτε οτι αυτή η μεταβλητή θα παίζει τον ρόλο του μετρητή οταν σαρώνουμε ένα απλό πίνακα.
Γραμμή 53:Η μεταβλητή – δείκτη λαμβάνει την αρχική τιμή HeadAddress η οποία είναι η αρχική διεύθυνση της δομής που θέλουμε να σαρώσουμε
Γραμμή 54: Η μεταβλητή i θα είναι η μεταβλητή-μετρητής η οποία θα σηματοδοτεί το τέλος της σάρωσης ...
Γραμμή 55: Επαναληπτική δομή for() η οποία θα τερματίσει μόνο όταν η i (Μεταβλητή-μετρητή) γίνει ίση με την τιμή της παραμέτρου position (Όταν φτάσουμε στον κόμβο που επιθυμούμε)
Γραμμή 56:Πρωχωράμε στον επόμενο κόμβο..
Γραμμή 58: Αφού η επαναληπτική διαδικασία έχει τελείωσει , τότε ο κόμβος που αναζητούμε είναι ο p (ο κόμβος που σαρώθηκε τελευταίος) , έτσι εκτυπώνουμε τα δεδομένα του

Αν όλα πάνε καλά...

και έχετε κάνει ολα αυτά τα οποία εξηγήσαμε , πίστά και χωρίς λάθη , τότε το αποτέλεσμα των παρακάτω εντολών....

```
78  int main()
79  {
80      Push(10);
81      Push(20);
82      Push(30);
83      Index(HeadNode,1);
84
85  }
86
```

Θα μας δίνει το παρακάτω αποτέλεσμα

```
atticadreamer@atticadreamer-desktop:~$ ./a.out
20
```

*Όσο αναφορά τις συντεταγμένες των κόμβων , η αρίθμηση είναι όμοια με αυτή των πινάκων (arrays) ,καθώς ξεκινάει απο το μηδέν για τον πρώτο κόμβο και ούτω κάθε εξής .

Όσο αναφορά τις μετατροπές...

Η index() δεν χρειάζεται ουδεμία μετατροπή , καθώς είναι συμβατή με όλα τα είδη γραμμικών λίστων.Εδώ να σημειωθεί οτι η εισαγωγή στην position αριθμού , μεγαλύτερου απο τον συνολικό αριθμό κόμβων , θα αναγκάσει την index() να βγεί σε περιοχές μνήμης που δεν ανήκουν στην λίστα μας,είναι κάτι το οποίο πρέπει να αποφεύγεται!.

Ανωμαλίες της Index()

Στις κυκλικά συνδεδεμένες λίστες , αν δωθεί ως παράμετρος , αριθμός μεγαλύτερος απο τον συνολικό αριθμό κόμβών,θα αναγκάσει την index() να ξαναμετράει απο την αρχή τους κόμβους μόλις υπέρβει και τον κόμβο ουράς !

- Συνάρτηση GetTop()
  - Σύνταξη : Int GetTop(struct NewNOde *HeadAddress)

Η συνάρτηση αυτή λαμβάνει μόνο μία παράμετρο , η οποία είναι η αρχική διεύθυνση της δομής μας (Δηλαδή την διεύθυνση HeadNone). Δουλεία της είναι να επιστρέφει τον τελεύταιο κόμβο που μπήκε στην δομή μας!(Τον Κόμβο κεφαλής)

Σημείωση!

Σε περίπτωση που ο κόμβος έχει δεδομένα αλλου τύπου(float,char,string,bool Κτλπ) τότε προφανώς και η συνάρτηση GetTop() θα πρεπει επιστρέφει αντίστοιχο τύπο δεδομένων. Για παράδειγμα αν η μεταβλητή data στην δομή NewNode είναι τύπου char τότε αντοιστοίχως η GetTop() θα πρέπει να επιστέφει δεδομένα char

Δημιουργώντας την GetTop()

Η GetTop() είναι μια πανεύκολη συνάρτηση , η οποία υλοποιείται σε μόλις μια γραμμή.Η απλή λειτουργία της περιορίζεται στο να επιστρέφει τα δεδομένα του κόμβου HeadNode!.

```
85
86  int GetTop(struct NewNode *HeadAddress){return HeadAddress->data;}
87
88
```

Όπως είναι προφανές , δεν υπάρχουν παραλλαγές ανάμεσα στους τύπους των διαφορετικών γραμμικών λίστων , και αυτό είναι ως αποτέλεσμα της απλότητας της συνάρτησης αυτής !

- Συνάρτηση IsEmpty()
  - Σύνταξη :bool IsEmpty()

Η συνάρτηση αυτή επιστρέφει Boolean Τιμή αληθείας σε περίπτωση που η δομή δεδομένων μας είναι κενή ή ψεύδους σε αντίθετη περίπτωση !
Η συνάρτηση IsEmpty() είναι ίσως , η πιο εύκολη συνάρτηση που θα μπορούσε να υπάρξει (και ναί! Είναι πιο εύκολη απο την GetTop()) . Η δουλεία της είναι , έπειτα απο την κλήση της, να μας ενημερώνει αν η λίστα μας είναι κενή ή όχι .!Για να γίνει αυτό εφικτό , θα πρέπει απλώς να ελένξουμε την HeadNode , και αν …..

- Είναι NULL : Τότε η λίστα μας είναι κενή
- Δεν είναι NULL : Τότε η λίστα μας δεν είναι κενή

Ας δούμε τον κώδικα της IsEmpty()

```
179  int IsEmpty(struct NewNode *HeadAddress) {
180      if (HeadAddress==NULL)return 1;
181      else return 0;
182  }
```

Όπως είναι προφανές , η συνάρτηση αυτή , λόγω της απλότητας της δεν έχει προβλήματα

συμβατότητας , ανάμεσα στα είδη των λιστών . Έτσι η IsEmpty() είναι έτοιμη για δράση!

Πιο ευκόλα δεν γίνεται;

Το tutorial αυτό δημιουργήθηκε με γνώμωνα να σας διδάξει πως λειτουργούν οι γραμμικές λίστες μέσω εκμάθησης της δημιουργίας τους. Η δομή αυτή μπορεί επίσης να δημιουργήθει μέσω έτοιμων βιβλιοθηκών με μόνο μία γραμμή κώδικα!

Vectors in c++: (Μια ιδέα απο το επόμενο tutorial)

Ενα Vector είναι ,με λίγα λόγια ,μια γραμμική λίστα η οποία υλοποιείται μέσω έτοιμων συναρτήσεων! Ας δούμε πως μπορούμε να συντάξουμε ενα Vector

```
1  #include <cstdio>
2
3  #include <vector>
4
5  using namespace std;
6
7  int main(){
8
9      vector<int> f; //Δημιουργία του vector
10
11     f.push_back(10); //εισαγωγή κόμβου με τιμή 10
12
13     f.pop_back(); //διαγραφή του πρώτου κόμβου(Κόμβος κεφαλής)
14
15 }
16
```

Είναι προφανες ότι είναι πολύ πιο εύκολο να υλοποιήσουμε μια γραμμική λίστα μέσω vectors , αυτό πρέπει να χρησιμοποιείται στην “*καθημερινή” ζωή ενός προγραμματιστή.Αλλα ενας καλός προγραμματιστής επίσης πρέπει να γνωρίζει πως λειτουργούν τα πράγματα σε πιο low-level επίπεδο.αν δεν το κάνει.πάντα θα μειονεκτεί έναντι άλλων*

AtticaDreamer Για το 101projects.eu.

Τέλος