Δυναμικές Δομές Δεδομένων
Δημιουργία στοίβας(LIFO) σε γλώσσα προγραμματισμού C

Καλως ορίσατε στο tutorial του 101projects.eu για δημιουργία στοίβας σε γλώσσα προγραμματισμού C . Στο tutorial αυτό,θα δημιουργήσουμε και θα κατανοήσουμε πλήρως μια δυναμική δομή δεδομένων στοίβας,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,θα είναι παιχνιδάκι,Όμως για την ευκολότερη κατανοήση του tutorial αυτού,συνίσταται η γνώση ΤΟΥΛΆΧΙΣΤΟΝ των παρακάτω δυνατοτήτων της γλώσσας προγραμματισμού C .

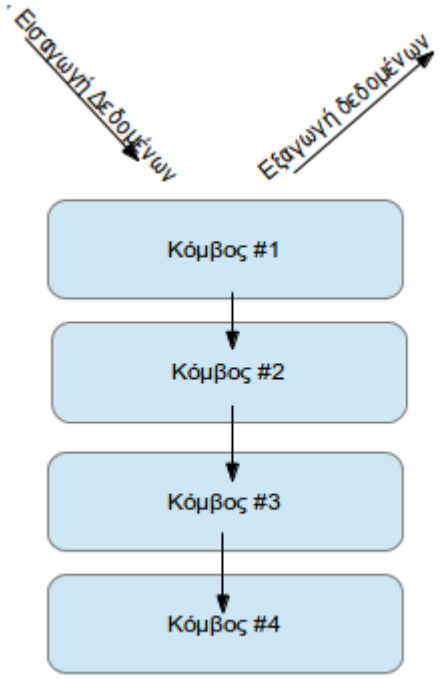
- Δομές (Structs)
- Δείκτες(Pointers) καθώς και την αριθμητική τους
- Δυναμική διαχείριση μνήμης με συναρτήσεις malloc() και free()
- Δημιουργία Συναρτήσεων τύπου-δείκτη με παραμέτρους δείκτες

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

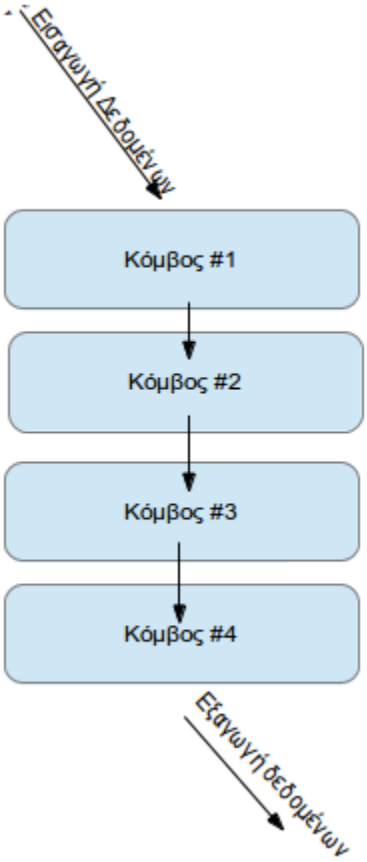
- Στοίβα

Η Στοίβα είναι μια απο τις πιο απλές δομές δεδομένων , ή οποία ακολουθεί την αρχιτεκτονική LIFO (Last In First Out) . Κάτα την αρχιτεκτονική αυτή , όπως υποδηλώνει το όνομα της ,κάθε κόμβος που προστίθεται , αφαιρείται πρώτος,όταν αυτό ζητηθεί



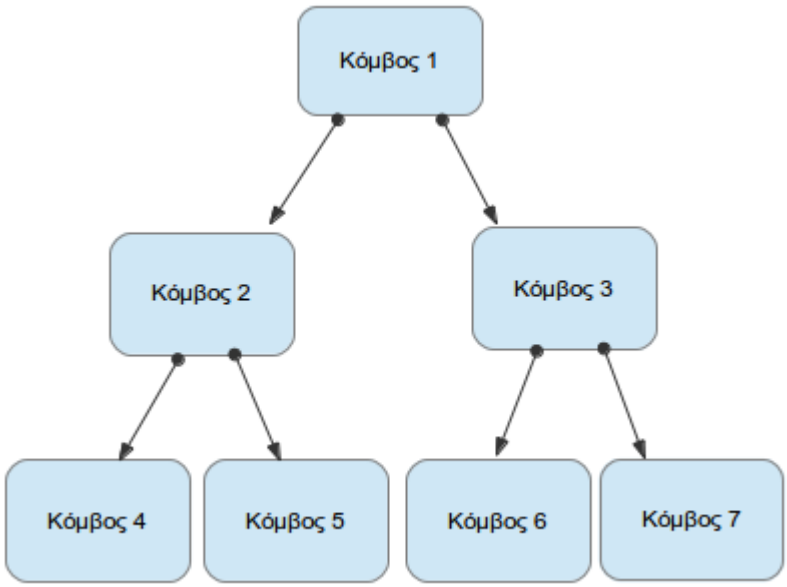
- Ουρά

Η ουρά είναι μια αρκετά απλή δομή δεδομένων η οποία ακολουθεί την αρχιτεκτονική FIFO.Κατα την αρχιτεκτονική αυτή , κάθε κόμβος που προστίθεται , αφαιρείται με την ίδια σείρα που μπήκε,ο πρώτος κόμβος που μπήκε στην δομή θα βγεί πρώτος,και ο τελευταίος θα βγεί τελευταίος.



- 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 !



- Γράφος(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;

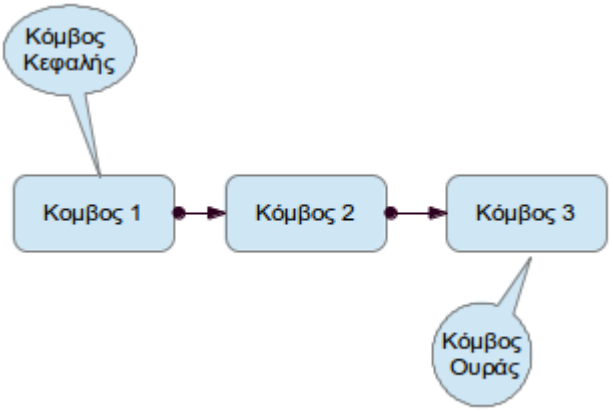
Όταν δηλώνουμε έναν πίνακα,καθορίζουμε απο την αρχή το μέγεθος του,χωρίς να μπορούμε να το μεταβάλλουμε στην συνέχεια .Ο πίνακας αυτός θα παραμείνει σταθερός καθ όλη την διάρκεια του προγράμματος.Αν εμείς ομώς δεν μπορούμε να προβλέψουμε το μέγεθος των δεδομένων που θα λαμβάνουμε; Τι συμβαίνει σε αυτή 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.

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 στοίβας μας ονομάζεται Head node (κόμβος κεφαλής) και ο τελευταίος ονομάζεται Tail node (Κόμβος Ουράς) .

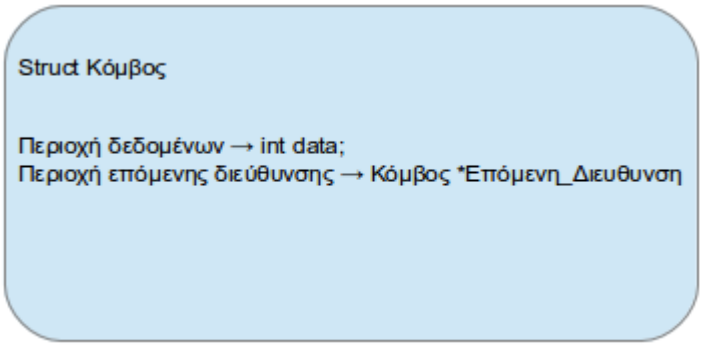


Τι ακριβώς είναι αυτός ο <<κόμβος>>;

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> //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 *Next; //περιοχή επόμενης διεύθυνσης του κόμβου
6
7
8  };//Τέλος Δήλωσης της δομής
9
```

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!

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 Next. 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 Next ως τύπου δείκτη -Void.αλλα κάτι τέτοιο πραγματικά δεν συνίσταται,για τον απλούστατο λόγο οτι οι δείκτες τύπου -Void δεν χρησιμοποιούν την αριθμητική των δεικτών,πράγμα που θα μπορούσε να αποβεί αιτία πολλών προβλημάτων για την δομή μας (Καταστροφή δεδομένων αλλων προγραμμάτων ,σφάλμα της δομής μας κτλπ)

Στο ρεζουμέ : Δημιουργώντας την στοίβα

Για την δημιουργία, και την διαχείριση της στοίβας μας , μας χρειάζονται συνολικά 6 βασικές συναρτήσεις , τις οποίες εμείς πρόκειται να δημιουργήσουμε

- Συνάρτηση : Push()
- Σύνταξη : Void Push (int NewNodeData,struct NewNode *HeadAddress)

Η συνάρτηση αυτή, θα δημιουργεί έναν καινούριο κόμβο και αφού θα του "περνάει" τα δεδομένα του θα τον προσαρμόζει κατάλληλα στην στοιβα μας συνδέοντας τον με τους υπόλοιπούς κόμβους αυτόματα.Η συνάρτηση αυτή θα λαμβάνει 1 μόνο παραμέτρου τύπου Integer.Η μεταβλητή NewNodeData θα περιέχει τα δεδομένα του κόμβου που πρόκειται να εισαχθεί στην δομή μας .

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 στοίβα χωρίζεται σε δύο υποκατηγορίες,ανάλογα με την περίπτωση

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

Εισαγωγή Κόμβου σε κενή στοίβα

Η διαδικασία αυτή είναι εξαιρετικά απλή , καθώς μόλις αρχικοποιούμε τον κόμβο , απλά τον δηλώνουμε ως κόμβο κεφαλής,αποθηκεύοντας την διεύθυνση του στον δείκτη HeadNode.

```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


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

```

Τι είναι η HeadNode?

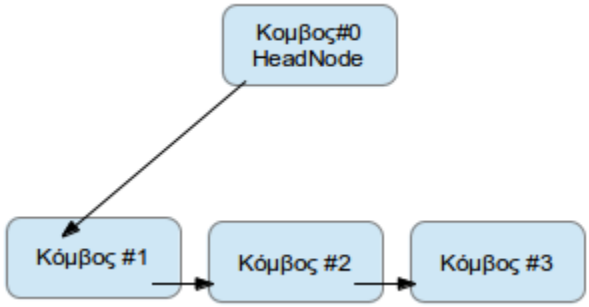
Πριν ξεκινήσουμε την Push(), **δηλώνουμε μια μεταβλητή-δείκτη τύπου NewNode με όνομα HeadNode**(Γραμμή 9) .Αυτός είναι ο δείκτης προς τον κόμβο κεφαλής.Αρχικά τον δηλώνουμε με τιμή NULL και πάντοτε εκτός κάπιας συνάρτησης για να είναι ορατός απο ολο το πρόγραμμα (Καθολική εμβέλεια).Απο την HeadNode θα έχουμε πρόσβαση σε όλη την στοίβα 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 στοίβα 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 στοίβα , έτσι γνωρίζουμε οτι αν ο κόμβος κεφαλής δεν περιέχει τίποτε τότε η στοίβα μας είναι κενή!

- Η HeadNode να είναι διάφορη του NULL

Πράγμα που σημαίνει προφανώς οτι η στοίβα 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 δομης δεδομένων 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.Για να <<μπούμε>> στην στοίβα μας , χρειάζεται να γνωρίζουμε την διεύθυνση κάπιου κόμβου. Για τον σκόπο αυτό, υπάρχει ο δείκτης HeadNode, καθώς αυτός θα μας γνωστοποιεί την αρχική διεύθυνση της στοίβας μας ,για να μπορούμε να την επεξεργαστούμε,όταν αυτό χρειαστεί.
Όπως θα παρατηρήσετε , αρκετές συναρτήσεις λαμβάνουν ως παράμετρο την Εναρκτήρια διεύθυνση της στοίβας μας (Καλή ώρα όπως η Push που αναλύουμε τώρα!)

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 στοίβα μας 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!

Ο κόμβος ουράς , καθώς δεν περιέχει επόμενο κόμβο απο αὐτόν,στην περιοχή επόμενης διεύθυνσης λαμβάνει την τιμή NULL

   - 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 στοίβα !

```
13  void Push(int NewNodeData)
14  {
15      struct NewNode *NodeToEnter = (struct NewNode *)malloc(sizeof(struct NewNode));
16
17
18      NodeToEnter->data=NewNodeData;
19
20      if (HeadNode==NULL)
21      {
22          HeadNode = NodeToEnter;
23          NodeToEnter->Next=NULL;
24
25      }
```

Εξηγώντας Γραμμή-Γραμμή τι κάναμε....

Γραμμή 13 : 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 18 : 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 20: Ελένχουμε αν η δομή δεδομένων μάς είναι κενή,εάν είναι τότε....
Γραμμή 22 : ...Απλά αποθηκεύουμε την διεύθυνση του νεού κόμβου στον δείκτη HeadNode,έτσι να τον δηλώσουμε ως τον πρώτο κόμβο!
Γραμμή 23 : 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 στοίβα

Ηρθε λοιπόν η στιγμή να ολοκληρώσουμε αυτή την συνάρτηση τελειοποιώντας την , έτσι ώστε να εισάγει αυτόματα κόμβους και σε προυπάρχουσα στοίβα

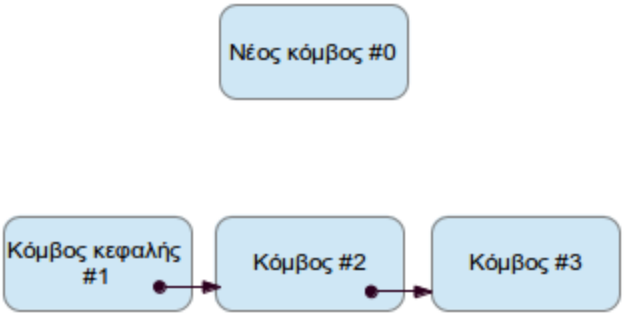
Αλγόριθμος εισαγωγής κόμβου σε προυπάρχουσα στοίβα

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

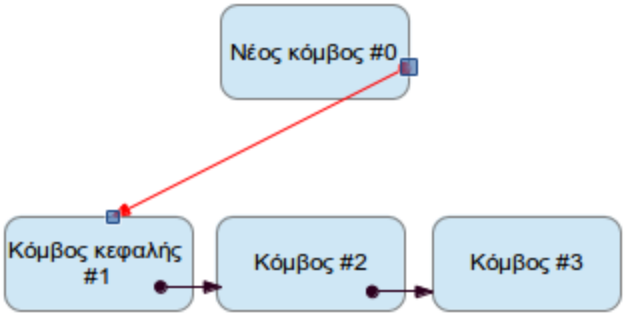
1. Δηλώνουμε την επόμενη διεύθυνση του νεόυ μας κόμβου τον υπάρχων κόμβο κεφαλής
2. Δηλώνουμε τον νεό μας κόμβο ως κόμβο κεφαλής

Ας το δούμε σχηματικά τι επρόκειτο να κάνουμε...

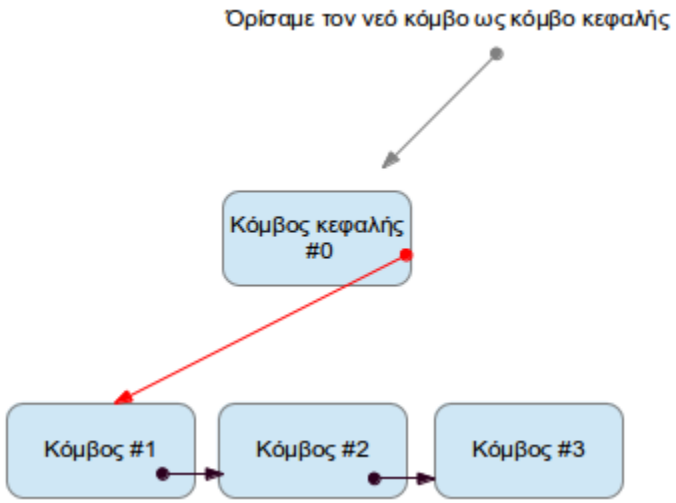
Έστω η παρακάτω στοίβα



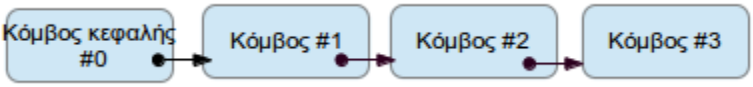
Πρώτον,θα αποθηκεύσουμε την διεύθυνση του κόμβου κεφαλής , στην περιοχή επόμενης διεύθυνσης του νέου μας κόμβου



Δεύτερον , θα αποθηκεύσουμε στον δείκτη κεφαλής HeadNode την διεύθυνση του νέου κόμβου



Η δομή δεδομένων μας έπειτα απο τα παραπάνω βήματα θα μοιάζει κάπως έτσι…



Ολοκληρώνωντας την Push()

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

```
13  void Push(int NewNodeData)
14  {
15      struct NewNode *NodeToEnter = (struct NewNode *)malloc(sizeof(struct NewNode));
16
17
18      NodeToEnter->data=NewNodeData;
19
20      if (HeadNode==NULL)
21      {
22          HeadNode = NodeToEnter;
23          NodeToEnter->Next=NULL;
24
25      }
26      else{
27          NodeToEnter->Next = HeadNode;
28          HeadNode = NodeToEnter;
29
30
31
32      }
33
34
35
36  }
```

Εξηγώντας Γραμμή-Γραμμή...

Έχοντας εξηγήσει τις γραμμές 13-24 πιο πάνω ας εξηγήσουμε γραμμή γραμμή πως καταφέραμε να

εισάγουμε τον νεό μας κόμβο στην προυπάρχουσα στοίβα μας!

Γραμμή 27:Σε περίπτωση που ο κόμβος δείκτης κεφαλής δεν είναι NULL τότε , αντιστοιχούμε στην επόμενη διεύθυνση του νεού κόμβου την διεύθυνση του υπάρχοντος κόμβου κεφαλής....
Γραμμή 28: και τέλος , 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 στοίβα μας!, Η Push() είναι έτοιμη για δράση!

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

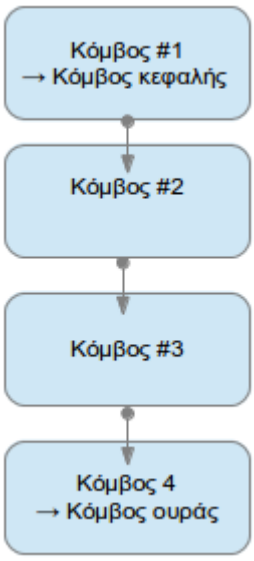
Η συνάρτηση αυτή δεν λάμβάνει παραμέτρους ,δουλεία της είναι , ύστερα απο την κλήση της ,να κάνει το ακριβώς ανάποδο απο οτι η Push! Θα διαγράφει κόμβους απο την στοίβα μας .

Δημιουργώντας την συνάρτηση pop()

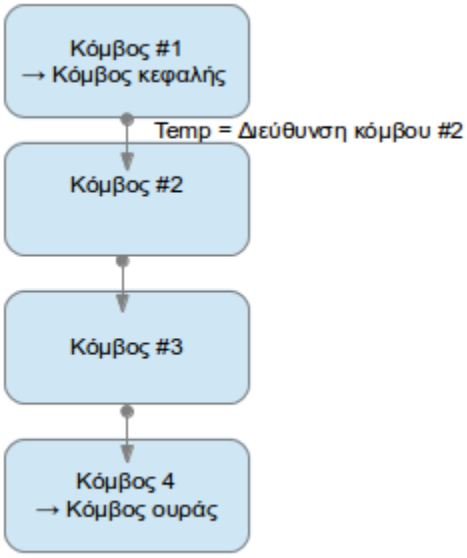
Θυμάστε την αρχιτεκτονική L.I.F.O την οποία υοθετεί η δομή δεδομένων στοίβας;Αυτό ακριβώς θα περιγράψουμε στις γραμμές που ακολουθούν.....
Η συνάρτηση Pop() είναι αρκετά εύκολη υπόθεση.η οποία χωρίζεται σε 3 απλά βήματα!

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

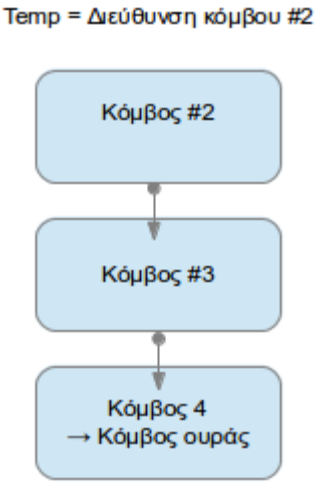
Ας δούμε σχηματικά τι εννοούμε
Έστω η δομή



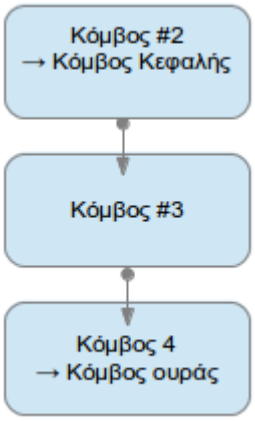
Βήμα 1) Αποθήκευση του επόμενου κόμβου απο τον κόμβο κεφαλής



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() , και ας εξηγήσουμε τι κάναμε γραμμή-γραμμή

```
71  void Pop()
72  {
73      struct NewNode *TheNextHeadNode = HeadNode->Next;
74      free(HeadNode);
75      HeadNode = TheNextHeadNode;
76  }
77
```

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 71 : Δήλωση της Pop ως μιας συνάρτησης τύπου void
Γραμμή 73:Αποθήκευση της διεύθυνσης του επόμενου κόμβου απο τον κόμβο κεφαλής (Βήμα 1)
Γραμμή 74: Διαγραφή του κόμβου κεφαλής με την χρήση της συνάρτησης free()(Βήμα 2)
Γραμμή 75 : Επαναπροσδιορισμός του κόμβου κεφαλής ως ο επόμενος κόμβος απο αυτόν που διαγράφτηκε , όπως προαναφέραμε στο βήμα 3

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

Η συνάρτηση αυτή λαμβάνει μόνο μια παράμετρο,η οποία είναι η εναρκτήρια διεύθυνση της στοίβας 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

```
38  void Display(struct NewNode *HeadAddress)
39  {
40      struct NewNode *p;
41      p = HeadAddress;
42      while (p->Next != NULL){
43          printf("%d\n",p->data);
44          p=p->Next;
45      }
46      printf("%d\n",p->data);
47  }
```

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 38 : Δηλώνουμε την συνάρτηση ως void με παράμετρο την *εναρκτήρια διεθυνση* της στοίβας μας!
Γραμμή 40: 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 41:Η μεταβλητή – δείκτη λαμβάνει την αρχική τιμή HeadAddress η οποία είναι η αρχική διεύθυνση της δομής που θέλουμε να εκτυπώσουμε
Γραμμή 42: 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 43:Εκτυπώνουμε τα δεδομένα του κόμβου που βρισκόμαστε
Γραμμή 44: Η p ισούται με την διεύθυνση του επόμενου κόμβου(Μέσω της μεταβλητής επόμενης διεύθυνσης) .ουσιαστικά προχωρούμε στον επόμενο κόμβο!
Γραμμη 46: 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....

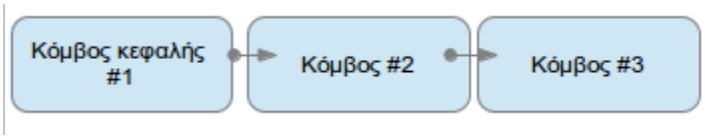
```
147 int main()
148 {
149     Push(10);
150     Push(20);
151     Push(30);
152     Display(HeadNode);
153 }
```

Θα έχει έξοδο ως εξής .....

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

- # 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 στοίβα



η κλήση της Index() με παραμέτρους την διεύθυνση του κόμβου κεφαλής , και τον αριθμό 1 . θα μας επιστρέψει τα δεδομένα του κόμβου #2 .Καθώς ξεκινάμε

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 στοίβα . 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

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

Εξηγώντας Γραμμή-Γραμμή...

Γραμμή 51: 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 52:Η μεταβλητή – δείκτη λαμβάνει την αρχική τιμή HeadAddress η οποία είναι η αρχική διεύθυνση της δομής που θέλουμε να σαρώσουμε
Γραμμή 53: Η μεταβλητή i θα είναι η μεταβλητή-μετρητής η οποία θα σηματοδοτεί το τέλος της σάρωσης ...
Γραμμή 54: 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 55:Πρωχωράμε στον επόμενο κόμβο..
Γραμμή 57: 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 .

- 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
```

- 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 η δεύτερη πιο εύκολη συνάρτηση που θα μπορούσε να υπάρξει (Με απόλυτο πρωτοπόρο την GetTop() Φυσικά) . Η δουλεία της είναι , έπειτα απο την κλήση της, να μας ενημερώνει αν η στοίβα μας είναι κενή ή όχι .Για να γίνει αυτό εφικτό , θα πρέπει απλώς να ελένξουμε την HeadNode , και αν …..

- Είναι NULL : Τότε η στοίβα μας είναι κενή
- Δεν είναι NULL : Τότε η στοίβα 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  }
```

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!.*Τέτοιες μεθόδους θα τις αναλυσουμε σε μεταγενέστερα tutorials*

AtticaDreamer Για το 101projects.eu.

Τέλος