Δομές Δεδομένων

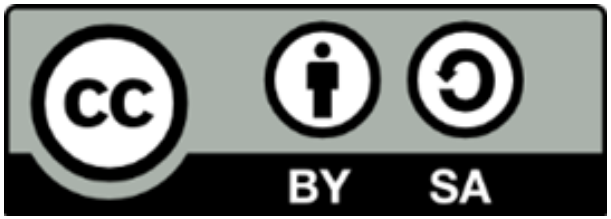
Ενότητα 5: Δείκτες και Δυναμική Δέσμευση-Αποδέσμευση Μνήμης στη C/ Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με δείκτες /Ένα πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα

Καθηγήτρια Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας[1]

- Να κατανοήσουν :
- την έννοια της μεταβλητής τύπου δείκτη
- Τη δήλωση μεταβλητής τύπου δείκτη
- Τις λειτουργίες που εκτελούν οι δείκτες και οι ανώνυμες μεταβλητές
- Η διαφορά μεταξύ της μεταβλητής τύπου δείκτη και της μεταβλητής στην οποία αναφέρεται ο δείκτης

Σκοποί ενότητας[2]

- Να μελετηθεί η Υλοποίηση της συνδεδεμένης λίστας με δείκτες καθώς και η υλοποίηση των πράξεων της Δημιουργίας & ελέγχου της κενής συνδεδεμένης λίστας και της Διάσχισης συνδεδεμένης λίστας.
- Να κατανοήσει την υλοποίηση του βασικών λειτουργιών της ΣΛ με τη χρήση δεικτών

Περιεχόμενα ενότητας[1]

- Εισαγωγή
- Οι συναρτήσεις malloc, free
- Δήλωση & παράδειγμα μεταβλητή δείκτη
- Ανώνυμες/δυναμικές μεταβλητές
- Η σταθερά δείκτη NULL
- Λειτουργίες που εκτελούνται με δείκτες
- Ανάθεση τιμής, σχεσιακοί τελεστές
- Εκφράσεις σύγκρισεις

Περιεχόμενα ενότητας[2]

- Υλοποίηση συνδεδεμένης λίστας με δείκτες
- Δημιουργία & έλεγχος κενής συνδεδεμένης λίστας
- Διάσχιση συνδεδεμένης λίστας

Περιεχόμενα ενότητας[3]

- Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες
- Δημιουργία
- Έλεγχος κενής ΣΛ
- Εισαγωγή στοιχείου
- Διαγραφή στοιχείου
- Διάσχιση
- Αναζήτηση στοιχείου σε ΣΛ

Δείκτες και Δυναμική Δέσμευση/ Αποδέσμευση Μνήμης στη C

Εισαγωγή -1-

- Από τον ορισμό της λίστας ως αφηρημένος τύπος δεδομένων προκύπτει ότι θεωρητικά μπορεί να εισαχθεί απεριόριστο πλήθος στοιχείων σ' αυτήν.
- Κατά συνέπεια, οποιαδήποτε υλοποίηση λίστας που χρησιμοποιεί πίνακα για την αποθήκευση των στοιχείων της δεν θα είναι πιστή αναπαράσταση λίστας, γιατί ένας πίνακας έχει σταθερό μέγεθος το οποίο δεν μπορεί να αλλάξει από τη στιγμή που θα ορισθεί και μετά.

Εισαγωγή -2-

- Μια πιο πιστή υλοποίηση λίστας θα πρέπει να έχει τη δυνατότητα δέσμευσης και αποδέσμευσης θέσεων μνήμης για τους κόμβους δυναμικά κατά τη διάρκεια εκτέλεσης του προγράμματος, χωρίς να χρειάζεται να είναι προκαθορισμένο το όριο μεγέθους της δεξαμενής κόμβων πριν την εκτέλεση του προγράμματος ή γενικώς πριν την εισαγωγή στοιχείου.
- Η δυνατότητα αυτή υπάρχει στην C και παρέχεται από τις προκαθορισμένες συναρτήσεις **malloc** και **free** (ορίζονται στην stdilb.h) που χρησιμοποιούνται σε συνδυασμό με τους τύπους δεδομένων.

Η συνάρτηση malloc

- Η συνάρτηση malloc χρησιμοποιείται για να δεσμεύει θέσεις μνήμης κατά τη διάρκεια εκτέλεσης του προγράμματος. Όταν κληθεί, επιστρέφει τη διεύθυνση μιας θέσης μνήμης στην οποία μπορεί να αποθηκευτεί μια τιμή.
- Για να αναφερόμαστε σ' αυτήν τη θέση μνήμης και να μπορούμε να αποθηκεύουμε δεδομένα και να τα ανακτούμε από αυτήν, χρησιμοποιούμε ένα ειδικό είδος μεταβλητής, που ονομάζεται μεταβλητή δείκτης ή απλά δείκτης και η τιμή της είναι η διεύθυνση μιας θέσης μνήμης.

Δήλωση μεταβλητής δείκτη

- Ο τύπος μιας μεταβλητής δείκτη που χρησιμοποιείται για αναφορά στη θέση μνήμης όπου είναι αποθηκευμένη κάποια τιμή πρέπει να οριστεί ως:

  type-identifier* identifier ή
  type-identifier *identifier

  όπου type-identifier είναι ο τύπος δεδομένων της τιμής που αποθηκεύεται.

  Πχ int* p; ή int *p;

- Ο δείκτης είναι δεσμευμένος σε αυτόν τον τύπο δεδομένων και δεν γίνεται να αποθηκευτούν δεδομένα άλλων τύπων στη θέση μνήμης στην οποία αναφέρεται.

Παράδειγμα μεταβλητής δείκτη -1-

- Για παράδειγμα, αν τα δεδομένα είναι ακέραιοι, τότε ένας δείκτης σε μια θέση μνήμης, που μπορεί να χρησιμοποιηθεί για να αποθηκευτεί ένας ακέραιος, μπορεί να δηλωθεί ως εξής:

  int *p1;

- Αυτή η μεταβλητή δείκτη p1 είναι δεσμευμένη στον τύπο int και μπορεί να χρησιμοποιηθεί μόνο για να αναφέρεται σε θέσεις μνήμης στις οποίες μπορούν να αποθηκευτούν τιμές αυτού του τύπου.
- Η p1 αναφέρεται ως μεταβλητή δείκτης προς int (ακέραιο) ή δείκτης προς int..

Παράδειγμα μεταβλητής δείκτη -2-

- Η συνάρτηση malloc μπορεί να χρησιμοποιηθεί για την απόκτηση μιας τέτοιας θέσης μνήμη κατά την εκτέλεση του προγράμματος. Η κλήση της συνάρτησης malloc είναι της μορφής:

  p1 = malloc(sizeof(int));

  και καταχωρεί τη διεύθυνση μιας θέσης μνήμης στον δείκτη p1.

  Επομένως, η εντολή p1 = malloc(sizeof(int));

  καταχωρεί μια διεύθυνση μνήμης, π.χ. 946, στη μεταβλητή p1.

Παράδειγμα μεταβλητής δείκτη -3-

- Δηλαδή ο δείκτης p1 κρατά την τιμή της θέσης μνήμης στην οποία είναι αποθηκευμένος ένας ακέραιος, και όχι ο ίδιος ο ακέραιος.
- Μάλιστα μπορεί αυτή η θέση μνήμης να είναι η θέση της πρώτης λέξης σε ένα μπλοκ διαδοχικών θέσεων μνήμης, όπως όταν ο δείκτης δείχνει σε εγγραφή, πίνακα κτλ

Ανώνυμες/δυναμικές μεταβλητές

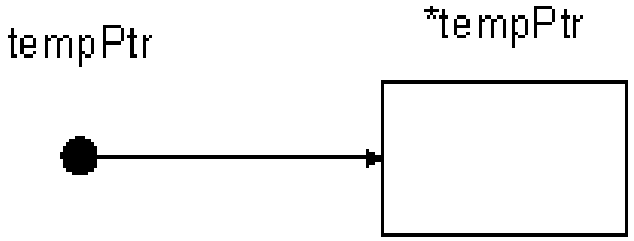
- Η προαναφερθείσα περιοχή μνήμης μπορεί να χρησιμοποιηθεί για να αποθηκεύει τιμές τύπου int, είναι μια μεταβλητή, η οποία όμως δεν έχει όνομα.
- Τέτοιες μεταβλητές ονομάζονται **ανώνυμες μεταβλητές (anonymous variables**) και οι δείκτες δείχνουν σε ανώνυμες μεταβλητές.
- Επειδή οι μεταβλητές αυτές υπάρχουν κατά τη διάρκεια εκτέλεσης του προγράμματος και μπορεί να πάψουν να υπάρχουν αργότερα, ονομάζονται και **δυναμικές μεταβλητές (dynamic variables)**.

Η σταθερά δείκτη NULL -1-

- Κάθε κλήση της malloc αποκτά μια νέα θέση μνήμης και καταχωρεί τη διεύθυνσή της στον συγκεκριμένο δείκτη.
- Έτσι, λοιπόν, αν η TempPtr είναι επίσης τύπου int, τότε η εντολή

  tempPtr = malloc(sizeof(int));

  αποκτά μια νέα θέση μνήμης στην οποία δείχνει ο TempPtr:



Η σταθερά δείκτη NULL -2-

- Η C παρέχει την **ειδική σταθερά δείκτη NULL** για την περίπτωση που θέλουμε να καταχωρήσουμε μια τιμή σε μια μεταβλητή δείκτη που δεν δείχνει σε καμιά θέση μνήμης.
- Αυτή η τιμή μπορεί να καταχωρηθεί σε δείκτη οποιουδήποτε τύπου με μια εντολή της μορφής: **pointer = NULL;**
- Όπως είναι αναμενόμενο ένας τέτοιος δείκτης θα συμβολίζεται απλά με μια τελεία:

  pointer ●

Λειτουργίες που εκτελούνται με τιμές δεικτών

- Οι τιμές των δεικτών είναι διευθύνσεις μνήμης, οι λειτουργίες που μπορούν να εκτελεστούν σ' αυτές :

- ανάθεση τιμής

- σύγκριση με τους σχεσιακούς τελεστές == και !=.

Ανάθεση τιμής -1-

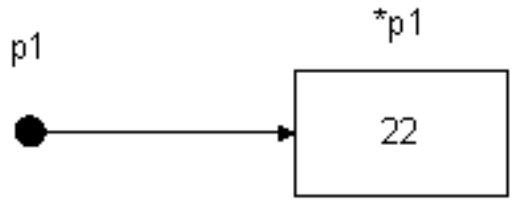
- Αν οι δείκτες ptr1 και ptr2 και είναι δεσμευμένοι στον ίδιο τύπο, τότε μια εντολή ανάθεσης τιμής της μορφής:

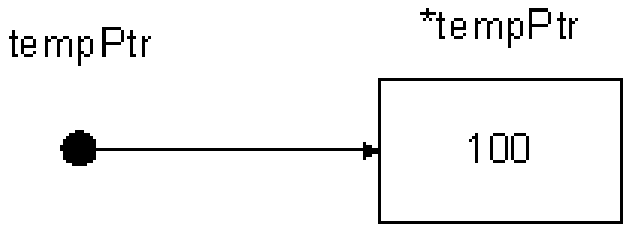
  ptr1 = ptr2;

- καταχωρεί την τιμή του ptr2 στον ptr1, οπότε και οι δυο δείχνουν στην ίδια θέση μνήμης.
- Η θέση στην οποία έδειχνε πρωτύτερα ο ptr1 δεν είναι πλέον προσπελάσιμη εκτός και αν κάποιος άλλος δείκτης δείχνει σ' αυτήν.

Ανάθεση τιμής -2-

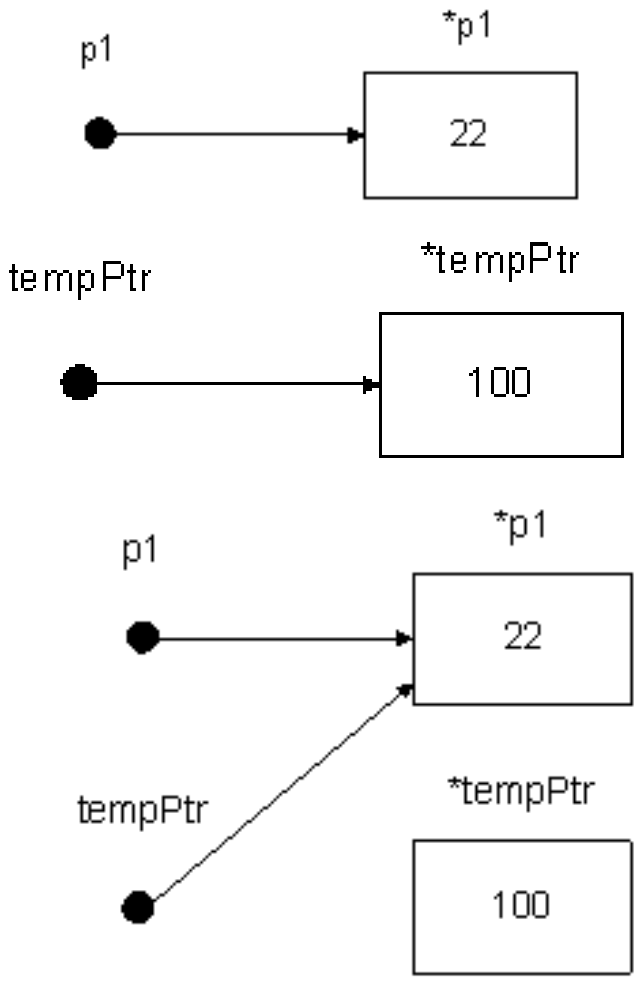
- Παράδειγμα: έστω ότι δυο δείκτες, οι p1 και tempPtr, είναι δηλωμένοι τύπου int και δείχνουν σε δυο θέσεις μνήμης που περιέχουν τις τιμές 22 και 100, αντίστοιχα.





Ανάθεση τιμής -3-

- Μια εντολή ανάθεσης
  tempPtr = p1 ;
- αναθέτει την τιμή της p1 στην tempPtr οπότε η tempPtr δείχνει στην θέση μνήμης που δείχνει και η p1.
- Τώρα δεν έχουμε πλέον πρόσβαση στη μεταβλητή *tempPtr, δηλαδή στο 100, εκτός και αν κάποιος άλλος δείκτης δείχνει εκεί.



Σχεσιακοί τελεστές

- Οι σχεσιακοί τελεστές == και != χρησιμοποιούνται για να συγκριθούν δυο δείκτες δεσμευμένοι στον ίδιο τύπο δεδομένων για να καθοριστεί αν και οι δύο δείχνουν στην ίδια θέση μνήμης ή αν είναι και οι δύο μηδενικοί.

  Η boolean έκφραση p1 **==** tempPtr

  είναι έγκυρη και είναι αληθής αν και μόνο αν οι p1 και tempPtr δείχνουν στην ίδια θέση μνήμης ή έχουν τιμή NULL.
- Επίσης, η έκφραση p1 **!=** NULL

  είναι κι αυτή μια έγκυρη boolean έκφραση.

Δείκτες ως παράμετροι/τιμές συνάρτησης

- Οι δείκτες μπορούν να χρησιμοποιηθούν και ως παράμετροι σε συναρτήσεις και διαδικασίες. Οι παράμετροι μπορεί να είναι είτε τιμές είτε μεταβλητές, όμως οι αντίστοιχοι δείκτες τυπικοί παράμετροι πρέπει να είναι δεσμευμένοι στον ίδιο τύπο. Ακόμα και η τιμή μιας συνάρτησης μπορεί να είναι ένας δείκτης.
- Για να αναφερθούμε στην τιμή που είναι αποθηκευμένη στη θέση μνήμης όπου δείχνει ένας δείκτης, χρησιμοποιούμε το σύμβολα * πριν από το όνομα του δείκτη: *pointer.

Ανάθεση τιμής -4-

- Έστω, για παράδειγμα, ότι έχουμε δηλώσει ένα δείκτη p1 τύπου int, ο οποίος δείχνει στη θέση μνήμης 2645 και θέλουμε να εκχωρήσουμε τη τιμή 22.

  Τότε:

  *p1 = 22

- Ένας μηδενικός ή μη ορισμένος δείκτης, όμως, δεν αναφέρεται σε καμιά θέση μνήμης, οπότε οποιαδήποτε προσπάθεια χρησιμοποίησής του είναι σφάλμα.

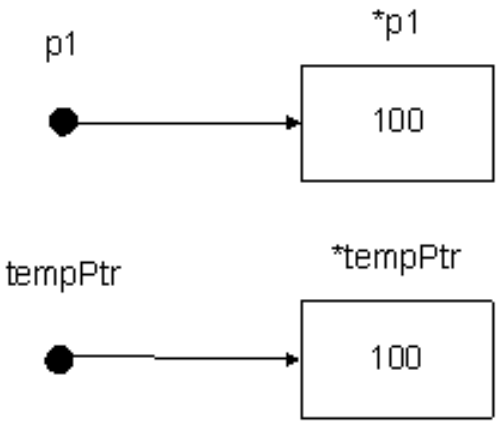
Ανάθεση τιμής -5-

- Αν οι p1 και tempPtr είναι μη μηδενικοί δείκτες τύπου int με

  *p1=22 και

  *tempPtr =100,

  τότε η εντολή *p1 = *tempPtr;



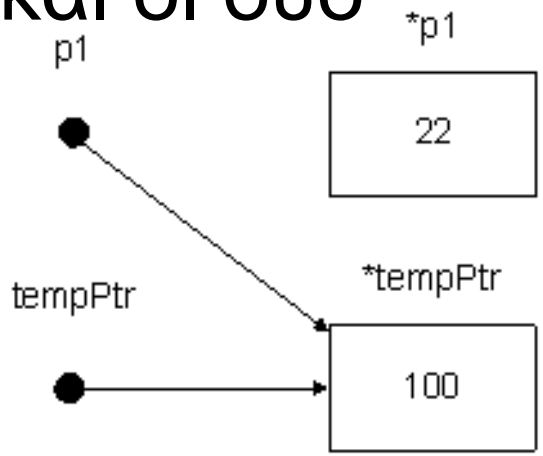
είναι μια έγκυρη εντολή ανάθεσης τιμής, αφού και οι δυο δυναμικές μεταβλητές *p1 και *tempPtr υπάρχουν και είναι του ίδιου τύπου. Η εντολή αυτή αντιγράφει το περιεχόμενο της θέσης μνήμης στην οποία δείχνει η p1 στην θέση μνήμης στην οποία δείχνει η tempPtr

Ανάθεση τιμής -6-

- Η παραπάνω εντολή διαφέρει πολύ από την

  p1 = tempPtr;

  η οποία δίνει την τιμή της p1 στην tempPtr με αποτέλεσμα να δείχνουν και οι δυο στην ίδια θέση μνήμης.



Εκφράσεις σύγκρισης -1-

- Επίσης, διαφορετική είναι η τιμή των boolean εκφράσεων σύγκρισης

  p1 == tempPtr; *p1 == *tempPtr;

  Η πρώτη έκφραση είναι αληθής αν και μόνο αν οι δείκτες p1 και tempPtr δείχνουν στην ίδια θέση μνήμης,

  ενώ η δεύτερη έκφραση συγκρίνει τον ακέραιο που είναι αποθηκευμένος στη θέση μνήμης στην οποία δείχνει ο p1 με τον ακέραιο που είναι αποθηκευμένος στη θέση μνήμης στην οποία δείχνει ο tempPtr.

Εκφράσεις σύγκρισης -2-

- Προφανώς:
  αν η p1 == tempPtr; είναι αληθής και
  οι δυο δείκτες δεν είναι μηδενικοί,
  τότε και η *p1 == *tempPtr; είναι αληθής.
- Το αντίστροφο όμως δεν ισχύει. Δηλαδή, αν *p1 == *tempPtr; δεν σημαίνει ότι οι p1 και tempPtr δείχνουν στην ίδια θέση μνήμης, αλλά απλά τα περιεχόμενά τους είναι ίδια.
- Τέλος, αν κάποιος από τους δύο δείκτες είναι μηδενικός, τότε η έκφραση p1 == tempPtr είναι έγκυρη, όχι όμως και η *p1 == *tempPtr δεν είναι έγκυρη.

Αποδέσμευση θέσης μνήμης

- Αν η θέση μνήμης στην οποία δείχνει ένας δείκτης δεν χρειάζεται πλέον, μπορεί να ελευθερωθεί και να γίνει διαθέσιμη για να δεσμευτεί αργότερα με κλήση της διαδικασίας free ως εξής:

```
free(pointer);
```

- Η διαδικασία αυτή ελευθερώνει τη θέση μνήμης στην οποία δείχνει η pointer και αφήνει τη μεταβλητή pointer μη ορισμένη.

Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

Υλοποίηση συνδεδεμένης λίστας με δείκτες -1-

- Στην υλοποίηση των συνδεδεμένων λιστών με πίνακα χρησιμοποιήσαμε εγγραφές ως τη βασική αποθηκευτική δομή για τους κόμβους.
- Έτσι και εδώ θα χρησιμοποιήσουμε πάλι εγγραφές με πεδία Data και Next.
- Ο τύπος δεδομένων του πεδίου Data θα είναι αυτός που είναι κατάλληλος για την αποθήκευση των δεδομένων και στο πεδίο Next θα αποθηκεύεται ένας δεσμός που θα δείχνει στο επόμενο στοιχείο της λίστας.
- Τώρα, όμως, ο δεσμός αυτός θα είναι ένας δείκτης και όχι ένας αριθμοδείκτης πίνακα..

Υλοποίηση συνδεδεμένης λίστας με δείκτες -2-

- Υλοποίηση συνδεδεμένης λίστας με δείκτες:

```
typedef int ListElementType;
    /*ο τύπος των στοιχείων της λίστας*/
typedef struct ListNode *ListPointer;
typedef struct ListNode
    {
            ListElementType Data;
            ListPointer Next;
    } ListNode;
```

- Ο ορισμός του αναγνωριστικού ListPointer προηγείται του ορισμού τού τύπου ListNode.

Δημιουργία & έλεγχος κενής συνδεδεμένης λίστας -1-

- Το πλεονέκτημα αυτής της υλοποίησης είναι ότι δεν χρειαζόμαστε μια δεξαμενή διαθέσιμων κόμβων, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις malloc και free, για απόκτηση και απελευθέρωση μνήμης αντίστοιχα, αντί για τις GetNode και ReleaseNode.

Δημιουργία & έλεγχος κενής συνδεδεμένης λίστας -2-

- Για τη δημιουργία μιας κενής συνδεδεμένης λίστας, αναθέτουμε απλά την τιμή NULL σε μια μεταβλητή List τύπου ListPointer,

  List = NULL;

  η οποία διατηρεί την πρόσβαση στον πρώτο κόμβο της συνδεδεμένης λίστας.

Δημιουργία & έλεγχος κενής συνδεδεμένης λίστας -3-

- Για να ελέγξουμε αν μια συνδεδεμένη λίστα είναι κενή, μπορούμε απλά να εξετάσουμε αν η List έχει τιμή NULL:

List == NULL;

Διάσχιση συνδεδεμένης λίστας

- Για τη διάσχιση μιας συνδεδεμένης λίστας υλοποιημένης με δείκτες, αρχικοποιούμε έναν δείκτη CurrPtr να δείχνει στον πρώτο κόμβο της συνδεδεμένης λίστας και στη συνέχεια παίρνει τις τιμές των δεσμών των κόμβων και διατρέχει τη λίστα.

```
CurrPtr = List;
while (CurrPtr != NULL) {
    /*Εδώ παρεμβάλλονται οι απαραίτητες εντολές για την επεξεργασία του CurrPtr->Data*/
     CurrPtr = CurrPtr->Next;
}
```

Ένα πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -1-

```
typedef int ListElementType;
   /*ο τύπος των στοιχείων της συνδεδεμένης λίστας*/
typedef struct ListNode *ListPointer;
   /*ο τύπος των δεικτών για τους κόμβους*/
typedef struct ListNode
{
        ListElementType Data;
                ListPointer Next;
} ListNode;
typedef enum {
                        FALSE, TRUE
} boolean;
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -2-

```
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List,
    ListElementType Item, ListPointer PredPtr);
void LinkedDelete(ListPointer *List, ListPointer
    PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List,
    ListElementType Item, ListPointer *PredPtr,
    boolean *Found);
void OrderedLimearSearch(ListPointer List,
    ListElementType Item, ListPointer *PredPtr,
    boolean *Found);
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -3-

```
void CreateList (ListPointer *List)
/*Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη
   λίστα.
  Επιστρέφει: Τον μηδενικό δείκτη List.*/
{
        List = NULL;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -4-

```
boolean EmptyList((ListPointer List)
/*Δέχεται: Μια συνδεδεμένη λίστα με τον List να
   δείχνει στον πρώτο κόμβο.
  Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα
   είναι κενή.
  Επιστρέφει: TRUE αν η λίστα είναι κενή και
   FALSE διαφορετικά.*/
{
   return List == NULL;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -5-

```
void LinkedTraverse(ListPointer List)
/*Δέχεται:  Μια συνδεδεμένη λίστα με τον List
   να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Διασχίζει τη συνδεδεμένη λίστα και
   επεξεργάζεται κάθε δεδομένο ακριβώς μια
   φορά.
  Επιστρέφει: Εξαρτάται από το είδος της
   επεξεργασίας.*/
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -6-

```
{
   ListPointer CurrPtr;
   CurrPtr = List;
   while (CurrPtr != NULL)
   {
        /*Εδώ παρεμβάλλονται οι απαραίτητες
        εντολές για την  επεξεργασία του
        CurrPtr->Data*/
        CurrPtr = CurrPtr->Next
   }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -7-

void LinearSearch (ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);

/*Δέχεται: Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.

Λειτουργία: Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη συνδεδεμένη λίστα για έναν κόμβο που να περιέχει το στοιχείο Item.

Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι TRUE, ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν η αναζήτηση δεν είναι επιτυχής η Found είναι FALSE.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -8-

```
{
   ListPointer CurrPtr = List;
   *PredPtr = NULL; *Found=FALSE;
    while (!(*Found) && CurrPtr != NULL )
        if (CurrPtr->Data==Item ) *Found=TRUE;
        else {
                *PredPtr = CurrPtr;
                CurrPtr = CurrPtr->Next;
        }
   }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -9-

void OrderedLimearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean *Found);

/*Δέχεται: Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα, που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία ο δείκτης List δείχνει στον πρώτο κόμβο.

Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης λίστας για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή για μια θέση για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο Item.

Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι TRUE, ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν η αναζήτηση δεν είναι επιτυχής η Found είναι FALSE.*/

.

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -10-

```
{
    ListPointer CurrPtr; boolean DoneSearching;
    CurrPtr = List; *PredPtr = NULL;
    DoneSearching = FALSE; *Found = FALSE;
    while (!DoneSearching && CurrPtr!=NULL ) {
        if (CurrPtr->Data >= Item {
                    DoneSearching = TRUE;
                    *Found = (CurrPtr->Data == Item); }
            else {
                     *PredPtr = CurrPtr;
                    CurrPtr = CurrPtr->Next; }
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -11-

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);

/*Δέχεται: Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο, ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.

Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη λίστα μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr ή στην αρχή της συνδεδεμένης λίστας, αν ο PredPtr είναι μηδενικός(NULL).

Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της να δεικτοδοτείται από τον List.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -12-

```
{
    ListPointer TempPtr;
   TempPtr = (ListPointer)malloc(sizeof(struct ListNode));
   TempPtr->Data = Item;
    if (PredPtr == NULL) {
          TempPtr->Next = *List;
          *List = TempPtr;
    }
    else {
                TempPtr->Next = PredPtr->Next;
                PredPtr->Next = TempPtr;
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -13-

```
void LinkedDelete(ListPointer *List, ListPointer
    PredPtr);
/*Δέχεται: Μια συνδεδεμένη λίστα με τον List να δείχνει
    στον πρώτο κόμβο της και έναν δείκτη PredPtr.
  Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον
    κόμβο που έχει για προηγούμενό του αυτόν στον οποίο
    δείχνει ο PredPtr ή διαγράφει τον πρώτο κόμβο, αν ο
    PredPtr είναι μηδενικός, εκτός και αν η λίστα είναι κενή.
  Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον
    πρώτο κόμβο να δεικτοδοτείται από τον List.
  Έξοδος: Ένα μήνυμα κενής λίστας αν η
    συνδεδεμένη λίστα ήταν κενή*/
.
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες -14-

```
{
    ListPointer TempPtr;
    if (EmptyList(*List)) printf("EMPTY LIST\n");
   else {
        if (PredPtr == NULL) {
                    TempPtr = *List;
                    *List = TempPtr->Next; }
        else {
                    TempPtr = PredPtr->Next;
                    PredPtr->Next = TempPtr->Next; }
        free(TempPtr);
    }
}
```

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης