



#### 4.4 Δείκτες και Δυναμική Δέσμευση/Αποδέσμευση

##### Μνήμης στη C

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 κατά τη διάρκεια εκτέλεσης του προγράμματος. 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. Η δυνατότητα αυτή υπάρχει στην C και παρέχεται από τις προκαθορισμένες συναρτήσεις malloc και free που χρησιμοποιούνται σε συνδυασμό με τους τύπους δεικτών.

Η συνάρτηση `void *malloc (int nBytes)` της C χρησιμοποιείται για να δεσμεύει τμήμα μνήμης ώστε να φιλοξενήσει το καθορισμένο πλήθος bytes κατά τη διάρκεια εκτέλεσης του προγράμματος. Όταν κληθεί, επιστρέφει έναν δείκτη προς τη πρώτη διεύθυνση αυτού του τμήματος μνήμης. 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 δείκτη* ή απλά *δείκτη* και η τιμή της είναι η διεύθυνση μιας θέσης μνήμης.

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. Ο δείκτης είναι δεσμευμένος σε αυτόν τον τύπο δεδομένων και δεν γίνεται να αποθηκευτούν δεδομένα άλλων τύπων στη θέση μνήμης στην οποία αναφέρεται. Για παράδειγμα, αν τα δεδομένα είναι `int`, 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.

Ενώ το πρόγραμμα εκτελείται, η συνάρτηση malloc μπορεί να χρησιμοποιηθεί για την απόκτηση μιας τέτοιας θέσης μνήμης. Η κλήση της συνάρτησης malloc είναι της μορφής, για παράδειγμα για έναν δείκτη προς int :

```
p1=malloc(sizeof(int));
```

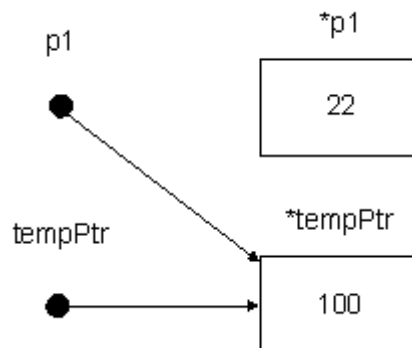
και καταχωρεί τη διεύθυνση μιας θέσης μνήμης στον δείκτη p1. Επομένως, η εντολή

```
p1=malloc(sizeof(int));
```

καταχωρεί μια διεύθυνση μνήμης, π.χ. 946, στον p1. Δηλαδή ο δείκτης p1 αποθηκεύει τη διεύθυνση της θέσης μνήμης \*p1 στην οποία είναι αποθηκευμένος ένας ακέραιος.



Λέμε ότι ο `p1` δείχνει σ' αυτήν τη θέση μνήμης και η γραφική αναπαράσταση που χρησιμοποιούμε είναι η εξής:

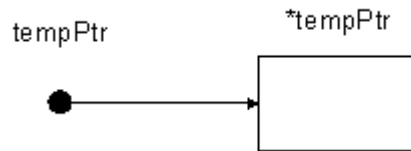


Εφόσον αυτή η περιοχή μνήμης μπορεί να χρησιμοποιηθεί για να αποθηκεύει τιμές τύπου `int`, είναι μια μεταβλητή, η οποία όμως δεν έχει όνομα. Τέτοιες μεταβλητές ονομάζονται **ανώνυμες μεταβλητές (anonymous variables)** και οι δείκτες μπορούν να δείχνουν σε ανώνυμες μεταβλητές. Επειδή αυτές οι μεταβλητές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

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

```
tempPtr = malloc(sizeof(int));;
```

αποκτά μια νέα θέση μνήμης στην οποία δείχνει ο tempPtr:



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 αναπαρίστανται απλά με μια τελεία:

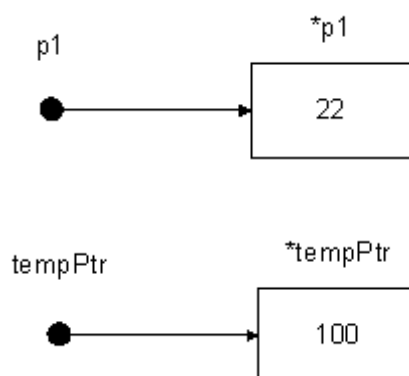
pointer •

Επειδή οι τιμές των δεικτών είναι διευθύνσεις μνήμης, οι λειτουργίες που μπορούν να εκτελεστούν σ' αυτές είναι οι: ανάθεση τιμής, σύγκριση με τους σχεσιακούς τελεστές == και !=. Αν οι δείκτες ptr1 και ptr2 και είναι δεσμευμένοι στον ίδιο τύπο, τότε μια πρόταση 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.

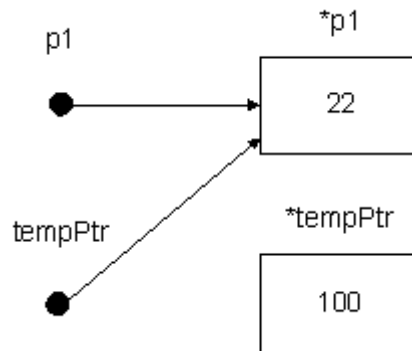
Έστω, για παράδειγμα, ότι δυο δείκτες, οι p1 και tempPtr, είναι δηλωμένοι δείκτες σε τύπου int και δείχνουν σε δυο θέσεις μνήμης που περιέχουν τις τιμές 22 και 100, αντίστοιχα.



Μια πρόταση 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, εκτός και αν κάποιος άλλος δείκτης δείχνει εκεί.

Οι σχεσιακοί τελεστές == και != μπορούν να χρησιμοποιηθούν 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. Επομένως, 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.

Για να αναφερθούμε στην τιμή που είναι αποθηκευμένη στη θέση μνήμης όπου δείχνει ένας δείκτης, χρησιμοποιούμε τα σύμβολα \* μετά από το όνομα του δείκτη:

```
*pointer
```

Έστω, για παράδειγμα, ότι έχουμε δηλώσει το p1 δείκτη προς τύπου int, και θέλουμε να εκχωρήσουμε τη τιμή 22. Τότε:

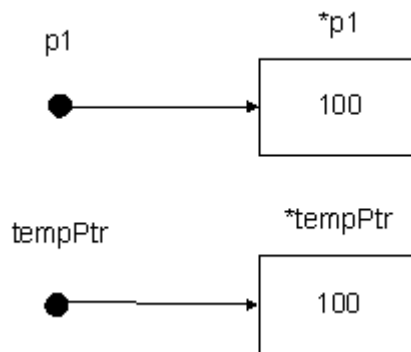
```
*p1 = 22
```

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

Αν οι `p1` και `tempPtr` είναι μη μηδενικοί δείκτες σε τύπου `int` με `*p1 = 22` και `*tempPtr = 100`, τότε η εντολή

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

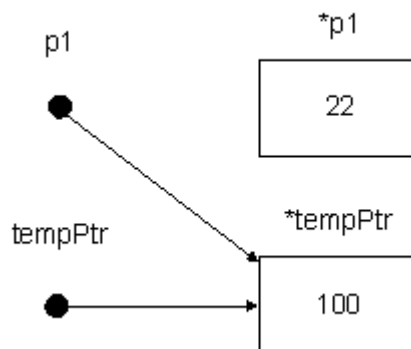
είναι μια έγκυρη εντολή ανάθεσης τιμής, αφού και οι δυο δυναμικές μεταβλητές `*p1` και `*tempPtr` υπάρχουν και είναι του ίδιου τύπου, `int`. Η εντολή αυτή αντιγράφει το περιεχόμενο της θέσης μνήμης στην οποία δείχνει ο `tempPtr`, δηλαδή το περιεχόμενο της `*tempPtr` στην θέση μνήμης στην οποία δείχνει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



Η παραπάνω πρόταση διαφέρει πολύ από την

```
p1 = tempPtr ;
```

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



Επίσης, διαφορετική είναι η τιμή των boolean εκφράσεων σύγκρισης

```
p1 == tempPtr
```

και

```
*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 τη τιμή που είναι αποθηκευμένη στη θέση μνήμης στην οποία δείχνει η `p1` με τη τιμή που είναι αποθηκευμένη στη θέση μνήμης στην οποία δείχνει η `tempPtr`. 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 ότι τα περιεχόμενά τους είναι ίδια.

Αν η θέση μνήμης στην οποία δείχνει ένας δείκτης δεν χρειάζεται πλέον, μπορεί να ελευθερωθεί με κλήση της συνάρτησης `free` και να γίνει διαθέσιμη για να δεσμευτεί αργότερα αν χρειασθεί. Η κλήση της `free` είναι:

```
free(pointer)
```

Η συνάρτηση αυτή ελευθερώνει τη θέση μνήμης στην οποία δείχνει η `pointer` και αφήνει την `pointer` μη ορισμένη.