

6. Δείκτες

Σύνοψη

Στο κεφάλαιο αυτό ο αναγνώστης εισάγεται στην έννοια του δείκτη. Αρχικά, δίνεται ο ορισμός και τα χαρακτηριστικά του δείκτη, η δήλωσή του και οι τρόποι ανάθεσης τιμής. Ακολούθως, μελετάται ο χειρισμός μεταβλητών και πινάκων με χρήση δεικτών. Στην επόμενη ενότητα παρουσιάζονται τα ζητήματα της κλήσης συνάρτησης κατ' αναφορά και των συναρτήσεων με επιστρεφόμενο τύπο δείκτη. Στη συνέχεια, περιγράφονται οι υλοποιήσεις συναρτήσεων αλφαριθμητικών με χρήση δεικτών. Το κεφάλαιο ολοκληρώνεται με τις έννοιες των ορισμάτων της γραμμής εντολών και του δείκτη σε συνάρτηση.

Λέξεις κλειδιά

δείκτης, κλήση κατ' αναφορά, διεύθυνση μεταβλητής και πίνακα, τελεστής διεύθυνσης, τελεστής περιεχομένου, συνάρτηση `strchr`, συνάρτηση `strstr`, ορίσματα γραμμής εντολών, δείκτης σε συνάρτηση.

Προαπαιτούμενη γνώση

Λεξιλόγιο της γλώσσας C – μεταβλητές – εκφράσεις – τελεστές – έλεγχος ροής προγράμματος – συναρτήσεις – πίνακες

6.1 Η έννοια του δείκτη

Σε κάθε μεταβλητή αποδίδεται μία θέση στην κύρια μνήμη του υπολογιστή, η οποία χαρακτηρίζεται από τη διεύθυνσή της. Στη γλώσσα μηχανής μπορεί να γίνει άμεση χρήση αυτής της διεύθυνσης, για να αποθηκευτούν ή να ανακληθούν δεδομένα. Αντίθετα, στις γλώσσες προγραμματισμού υψηλού επιπέδου οι διευθύνσεις δεν είναι άμεσα ορατές από τον προγραμματιστή, καθώς καλύπτονται από τον μανδύα των συμβολικών ονομάτων, τα οποία το σύστημα αντιστοιχεί στις πραγματικές διευθύνσεις.

Η γλώσσα C, θέλοντας να δώσει στον προγραμματιστή τη δυνατότητα συγγραφής αποδοτικού κώδικα, υποστηρίζει την άμεση διαχείριση των περιεχομένων της μνήμης εισάγοντας την έννοια του **δείκτη** (pointer). **Ο δείκτης αποτελεί μία ιδιόζουσα μορφή μεταβλητής, η οποία έχει ως περιεχόμενο όχι ένα πραγματικό δεδομένο αλλά μία διεύθυνση μνήμης.** Οι δείκτες είναι ένα ισχυρό προγραμματιστικό εργαλείο και εφαρμόζονται:

- στη δυναμική εκχώρηση μνήμης,
- στη διαχείριση σύνθετων δομών δεδομένων,
- στην αλλαγή τιμών που έχουν εκχωρηθεί ως ορίσματα σε συναρτήσεις,
- για τον αποτελεσματικότερο χειρισμό πινάκων.

Ωστόσο, καθώς η χρήση των δεικτών οδηγεί σε επεμβάσεις στη μνήμη και πολλές φορές σε προγραμματιστικές ακροβασίες, συχνά αποτελεί αιτία δύσκολων στον εντοπισμό σφαλμάτων, γι' αυτό και πρέπει να γίνεται με ιδιαίτερη προσοχή.

6.2 Δήλωση δείκτη

Η δήλωση ενός δείκτη ακολουθεί τον εξής formalισμό:

<τύπος δεδομένων> *<όνομα δείκτη>;

π.χ.

```
int *pnum;
```

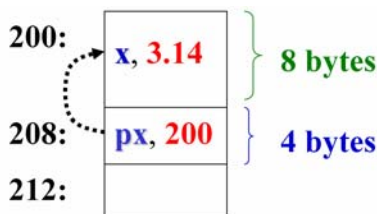
Όταν ένας δείκτης έχει αποθηκευμένη μία διεύθυνση, έχει επικρατήσει να λέγεται ότι ο δείκτης «δείχνει» στη διεύθυνση. Στη δήλωση δείκτη ο **<τύπος δεδομένων>** αφορά στο είδος των δεδομένων που αποθηκεύονται στη διεύθυνση που «δείχνει» ο δείκτης. Ο τύπος της κανονικής μεταβλητής πρέπει να δηλώνεται, γιατί μία μεταβλητή δεσμεύει συγκεκριμένη μνήμη (8 bytes για **double**, 4 bytes για **int** κ.ο.κ.). Εφόσον ο δείκτης χρησιμοποιείται για να γίνεται έμμεση αναφορά στην τιμή της μεταβλητής, πρέπει να είναι γνωστό πόση μνήμη καταλαμβάνει αυτή η τιμή.

Τον τύπο δεδομένων ακολουθεί ο αστερίσκος, ο οποίος είναι απαραίτητος, γιατί προσδιορίζει ότι δηλώνεται ένας δείκτης κι όχι μία κανονική μεταβλητή. Ο αστερίσκος συνδέεται με το όνομα και όχι με τον τύπο δεδομένων. Θα πρέπει να σημειωθεί ότι αν και ο δείκτης περιέχει μία διεύθυνση (δηλαδή έναν ακέραιο αριθμό), δεν είναι ίδιος με μία κανονική ακέραια μεταβλητή και δεν ακολουθεί την αριθμητική ακεραίων. Μέσω του αστερίσκου ο μεταγλωττιστής γνωρίζει ότι η τιμή του δείκτη είναι μία συγκεκριμένη διεύθυνση μνήμης, σε αντιδιαστολή με την «κανονική» ακέραια τιμή.

Τη δήλωση δείκτη κλείνει το όνομα του δείκτη. Επιλέγεται με βάση τις ίδιες συμβάσεις που ισχύουν στις κανονικές μεταβλητές. Ωστόσο, συνήθως ο αρχικός χαρακτήρας του ονόματος δείκτη είναι το **p**, έτσι ώστε το πρόγραμμα να καθίσταται περισσότερο ευανάγνωστο, καθώς με τον πρώτο χαρακτήρα φαίνεται εάν μία μεταβλητή είναι δείκτης ή όχι. Εναλλακτικά, μπορεί να προστεθεί η κατάληξη **_ptr**. Ενδεικτικές είναι οι ακόλουθες δηλώσεις δεικτών σε ακέραιες μεταβλητές και σε μεταβλητές τύπου χαρακτήρα:

```
int *pcount, *count_ptr;  
char *pword, *word_ptr;
```

Καθώς το περιεχόμενο ενός δείκτη είναι μία διεύθυνση, δηλαδή ένας ακέραιος αριθμός, ένας δείκτης θα καταλαμβάνει όσα bytes αντιστοιχούν σε ακέραιο (π.χ. 4 bytes), *ανεξάρτητα από τον τύπο της κανονικής μεταβλητής στην οποία δείχνει*. Στο **Σχήμα 6.1** απεικονίζεται ο τρόπος λειτουργίας των δεικτών, με τη **x** να είναι μία κανονική μεταβλητή τύπου **double** και τον **px** να είναι δείκτης που δείχνει στη **x**.



Σχήμα 6.1 Απεικόνιση του τρόπου λειτουργίας των δεικτών

6.3 Ανάθεση τιμής σε δείκτη

Η ανάθεση τιμής σε δείκτη μπορεί να γίνει με έναν από τους ακόλουθους τέσσερις τρόπους:

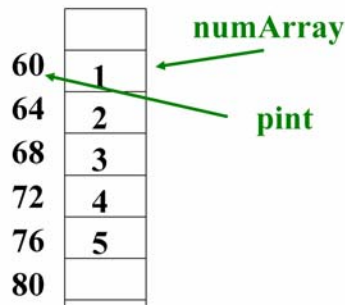
(1) **Με χρήση του τελεστή διεύθυνσης (&)** (address-of operator). Ο τελεστής διεύθυνσης **&**, ο οποίος πρωτοαπαντήθηκε στη συνάρτηση εισόδου **scanf()**, εισάγεται μπροστά από μία μεταβλητή εκφράζοντας τη διεύθυνσή της. Ο συμβολισμός **&count** ερμηνεύεται «στη διεύθυνση της **count**». Έτσι, με τον ακόλουθο κώδικα ανατίθεται στον δείκτη **pnum** η διεύθυνση, στην οποία βρίσκεται αποθηκευμένη η ακέραια μεταβλητή **count**.

```
int *pnum;  
int count;  
pnum=&count;
```

(2) **Με χρήση πινάκων**, δεδομένου ότι το όνομα ενός πίνακα αντιστοιχεί στη διεύθυνση του πρώτου στοιχείου του. Έτσι, με τον ακόλουθο κώδικα αποδίδεται στον δείκτη ακεραίων **pint** η τιμή **60**, δηλαδή η

διεύθυνση του πρώτου στοιχείου του πίνακα `numArray`. Στο Σχήμα 6.2 απεικονίζεται η διαδικασία ανάθεσης τιμής στον δείκτη `pint`.

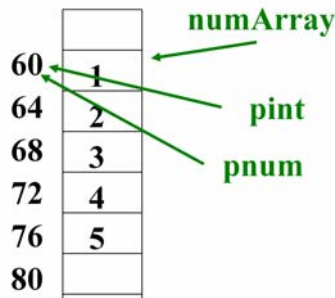
```
int numArray[5]={1,2,3,4,5};
int *pint;
pint=numArray;
```



Σχήμα 6.2 Ανάθεση τιμής σε δείκτη με χρήση πίνακα

(3) *Με χρήση άλλων δεικτών ίδιου τύπου.* Στον κώδικα που ακολουθεί, ο δείκτης `pint` δείχνει ήδη σε κάποια διεύθυνση. Με την έκφραση `pnum=pint;` το περιεχόμενο του `pint` αντιγράφεται στον `pnum` κι έτσι ο τελευταίος δείχνει στην ίδια διεύθυνση, όπως φαίνεται στο Σχήμα 6.3.

```
int numArray[5]={1,2,3,4,5};
int *pint,*pnum;
pint=numArray;
pnum=pint;
```



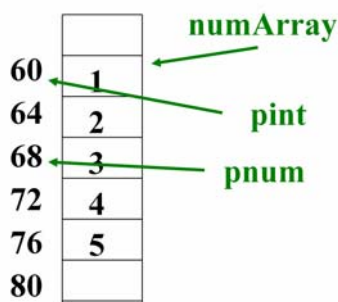
Σχήμα 6.3 Ανάθεση τιμής σε δείκτη με χρήση δεικτών ίδιου τύπου

(4) *Με χρήση αριθμητικής δεικτών.* Οι δείκτες υποστηρίζουν εκφράσεις της μορφής:

```
pnum=pint+y; ή pnum=pint-y;
```

όπου οι `pnum`, `pint` είναι δείκτες ίδιου τύπου και ο `y` είναι ακέραιος. Οι παραπάνω είναι οι μόνες πράξεις που μπορούν να γίνουν με δείκτες. Η διαφοροποίηση της πρώτης έκφρασης έγκειται στο ότι στον `pnum` δεν θα ανατεθεί το περιεχόμενο του `pint` αυξημένο κατά `y` μονάδες, αλλά αυξημένο κατά `y` επί τον αριθμό των bytes που καταλαμβάνει ο τύπος δεδομένων του δείκτη. Δηλαδή, εάν οι `pnum`, `pint` είναι δείκτες ακεραίων και το `y=2`, ο `pnum` θα δείχνει $2 \times 4 = 8$ bytes κάτω από τη διεύθυνση που δείχνει ο `pint`, όπως φαίνεται στο Σχήμα 6.4.

```
int numArray[5]={1,2,3,4,5};
int *pint,*pnum;
pint=numArray;
pnum=pint+2;
```



Σχήμα 6.4 Ανάθεση τιμής σε δείκτη με χρήση αριθμητικής δεικτών

Παρατήρηση: Θεωρητικά, μπορεί να γίνει άμεση ανάθεση μίας διεύθυνσης, π.χ. `pnum=1001;`. Ωστόσο, μία τέτοια επιλογή είναι πολύ επικίνδυνη και πρέπει πάντοτε να αποφεύγεται, καθώς κατά την εκτέλεση της γραμμής κώδικα δεν μπορεί να είναι γνωστό κατά πόσον οι διευθύνσεις από **1001** έως **1004** (δηλαδή τα 4 bytes που καταλαμβάνει ο ακέραιος, στον οποίο θα δείχνει ο `pnum`) είναι κατειλημμένες. Τέτοιου είδους αναθέσεις χρησιμοποιούνται συνήθως μόνο για άμεση πρόσβαση στο υλικό (hardware).

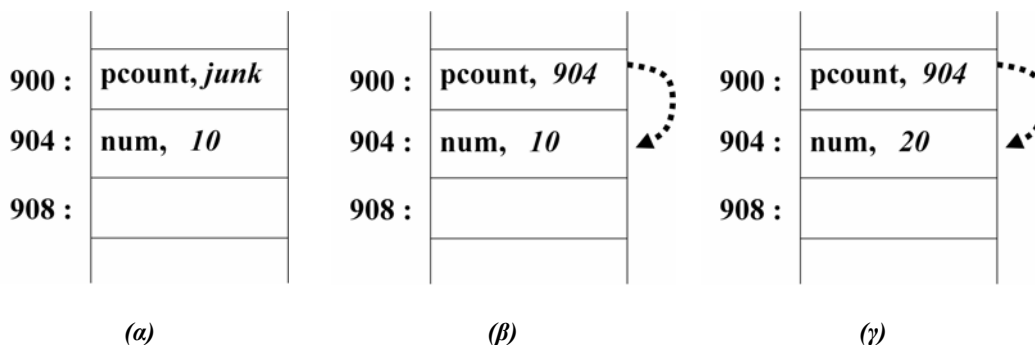
6.4 Προσπέλαση μεταβλητής και πίνακα με χρήση δείκτη

Η δεικτοποίηση μίας μεταβλητής μέσω δείκτη γίνεται με χρήση του **τελεστή περιεχομένου** (`*`) (dereferencing operator) ή **τελεστή έμμεσης αναφοράς** ή **έμμεσης διευθυνσιοδότησης**, η λειτουργία του οποίου περιγράφεται με τη βοήθεια του ακόλουθου κώδικα:

```
int *pcount, num;
num=10;
pcount=&num;
*pcount=20;
```

Στον παραπάνω κώδικα ορίζεται μία ακέραια μεταβλητή `num`, η οποία λαμβάνει την τιμή **10**, και ένας δείκτης σε ακέραιο `pcount`, ο οποίος αρχικά δεν έχει τιμή. Στο Σχήμα 6.5.α παρουσιάζεται ο χάρτης μνήμης μετά το τέλος της δεύτερης γραμμής κώδικα. Ως *junk* (απορρίματα) συμβολίζεται το περιεχόμενο μίας θέσης μνήμης, όταν αυτό δεν έχει καθοριστεί από το πρόγραμμα.

Ακολούθως, ανατίθεται στον δείκτη `pcount` η διεύθυνση της `num` (Σχήμα 6.5.β). Στην τελευταία γραμμή κώδικα χρησιμοποιείται ο τελεστής περιεχομένου μπροστά από τον δείκτη. Ο συμβολισμός `*pcount` ερμηνεύεται «στη διεύθυνση που δείχνει ο `pcount`». Έτσι, η τελευταία γραμμή κώδικα ερμηνεύεται «να τοποθετηθεί το **20** στη διεύθυνση που δείχνει ο `pcount`», δηλαδή να μεταβληθεί έμμεσα η τιμή της `num` από **10** σε **20** (Σχήμα 6.5.γ).



Σχήμα 6.5 Προσπέλαση μεταβλητής με χρήση δείκτη

Οι δείκτες χρησιμοποιούνται και για την προσπέλαση πινάκων. Μάλιστα, στο επόμενο κεφάλαιο, όπου θα μελετηθεί η δυναμική διαχείριση μνήμης, η δομή του πίνακα θα υλοποιηθεί αποκλειστικά με χρήση δείκτη. Έστω ένας πίνακας ακεραίων `arr[5]`, ο οποίος αποθηκεύεται στη μνήμη στα bytes **1000** έως και **1019**, και ο δείκτης σε ακέραιο `parr`. Με την πρόταση `parr=arr;` ο δείκτης `parr` μπορεί πλέον να προσπελάσει τα δεδομένα του πίνακα, καθώς το όνομα ενός πίνακα αντιστοιχεί στο πρώτο byte της θέσης μνήμης που καταλαμβάνει το πρώτο στοιχείο του πίνακα. Με την ακόλουθη σημειογραφία:

`parr[i]=...` ή `*(parr+i)=...`

ο δείκτης αναθέτει με έμμεσο τρόπο τιμή (ή γενικά χειρίζεται) στο στοιχείο `arr[i]` του πίνακα `arr`. Κατά συνέπεια, μακροσκοπικά ο χειρισμός των στοιχείων του πίνακα με χρήση δείκτη φαίνεται απολύτως αντίστοιχος με τον χειρισμό μέσω μεταβλητής πίνακα. Η διαφορά έγκειται στο ότι ο δείκτης λειτουργεί έμμεσα, προσπελαύνοντας την *i*-στη θέση μνήμης (στο συγκεκριμένο παράδειγμα και με βάση την αριθμητική δεικτών) τα bytes **1000+(4*i)** έως **1000+(4*i)+3**, ενώ η μεταβλητή πίνακα προσπελαύνει άμεσα το στοιχείο `arr[i]`.

Η έκφραση `parr=arr;` είναι ισοδύναμη με την έκφραση `parr=&arr[0];`.

6.4.1 Παράδειγμα

Να περιγραφεί αναλυτικά η λειτουργία κάθε γραμμής κώδικα και να απεικονιστούν τα περιεχόμενα των θέσεων μνήμης που καταλαμβάνουν οι μεταβλητές.

```
1 int *px, *py, x=1, y=0;
2 int a[5]={2,4,5,6,7};
3 int i;
4 px=&x;
5 py=a;
6 for (i=0;i<5;i++)      *(py+i)=2*i;
7 y=*px;
8 py=&y;
9 px=a+3;
10 *px=21;
11 *py=*px+9;
12 x=*( &y );
```

Γραμμή 1: Δήλωση δύο δεικτών σε ακέραιο (`px`, `py`), ακολουθούμενη από δήλωση και αρχικοποίηση δύο ακεραίων μεταβλητών (`x`, `y`).

Γραμμή 2: Δήλωση πίνακα ακεραίων πέντε στοιχείων (`a`) και αρχικοποίησή αυτού.

Γραμμή 3: Δήλωση ακεραίας μεταβλητής (`i`) (Σχήμα 6.6.α).

Γραμμή 4: Ανάθεση της διεύθυνσης της μεταβλητής `x` στον δείκτη `px`, δηλαδή το περιεχόμενο του `px` είναι η διεύθυνση – το πρώτο byte – του `x` (Σχήμα 6.6.β).

Γραμμή 5: Ανάθεση της διεύθυνσης του πρώτου στοιχείου του πίνακα `a` στον δείκτη `py`, δηλαδή το περιεχόμενο του `py` είναι η διεύθυνση του `a[0]` (Σχήμα 6.6.γ).

Γραμμή 6: Επαναληπτική έκφραση, σε κάθε επανάληψη της οποίας λαμβάνουν χώρα τα ακόλουθα: η τιμή `2*i` τοποθετείται σε θέση μνήμης, η οποία βρίσκεται σε διεύθυνση `2*i` bytes μετά τη διεύθυνση που δείχνει ο `py` (Σχήματα 6.6.δ–6.6.η). Για παράδειγμα, εάν `i=3`, η τιμή **6** αποθηκεύεται στη θέση μνήμης `py+i=1245028+3*4=1245028+12=1245040`, θέση που καταλαμβάνει το στοιχείο `a[3]`. Με αυτόν τον τρόπο αποδίδεται στο `a[3]` η τιμή 6.

Απεικόνιση της μνήμης

διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 2
1245032	a[1], 4
1245036	a[2], 5
1245040	a[3], 6
1245044	a[4], 7
1245048	i, junk
1245052	py, junk
1245056	px, junk
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.α

Απεικόνιση της μνήμης

διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 2
1245032	a[1], 4
1245036	a[2], 5
1245040	a[3], 6
1245044	a[4], 7
1245048	i, junk
1245052	py, junk
1245056	px, 1245064
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.β

Απεικόνιση της μνήμης

διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 2
1245032	a[1], 4
1245036	a[2], 5
1245040	a[3], 6
1245044	a[4], 7
1245048	i, junk
1245052	py, 1245028
1245056	px, 1245064
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.γ

Απεικόνιση της μνήμης

διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 0 (i=0)
1245032	a[1], 4
1245036	a[2], 5
1245040	a[3], 6
1245044	a[4], 7
1245048	i
1245052	py, 1245028
1245056	px, 1245064
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.δ

Απεικόνιση της μνήμης

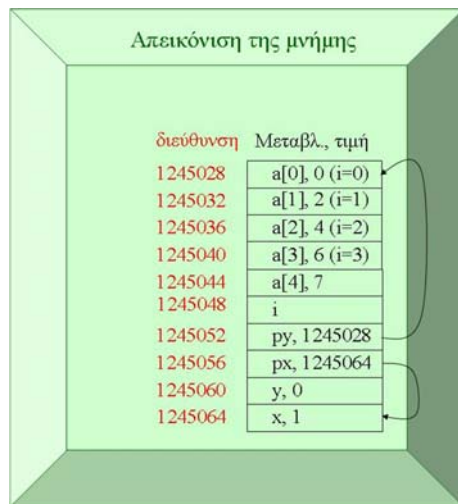
διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 0 (i=0)
1245032	a[1], 2 (i=1)
1245036	a[2], 5
1245040	a[3], 6
1245044	a[4], 7
1245048	i
1245052	py, 1245028
1245056	px, 1245064
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.ε

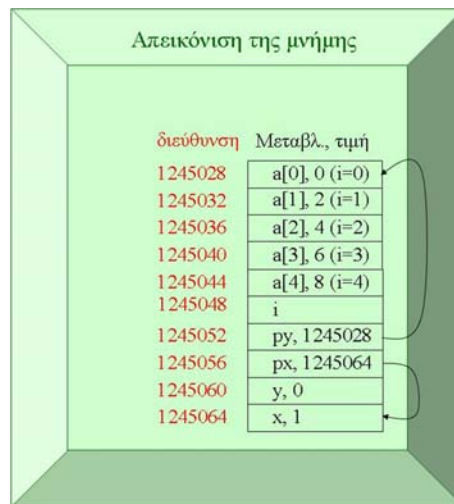
Απεικόνιση της μνήμης

διεύθυνση	Μεταβλ., τιμή
1245028	a[0], 0 (i=0)
1245032	a[1], 2 (i=1)
1245036	a[2], 4 (i=2)
1245040	a[3], 6
1245044	a[4], 7
1245048	i
1245052	py, 1245028
1245056	px, 1245064
1245060	y, 0
1245064	x, 1

Σχήμα 6.6.στ



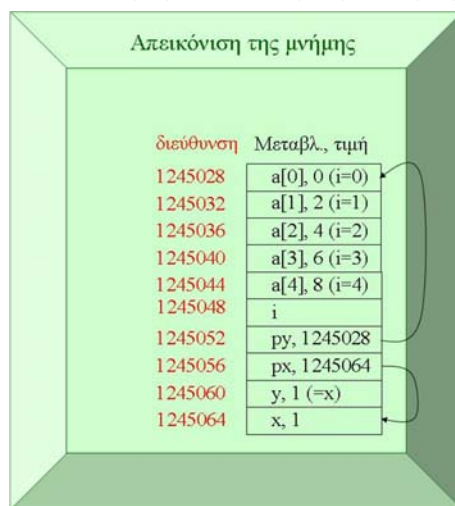
Σχήμα 6.6.ζ



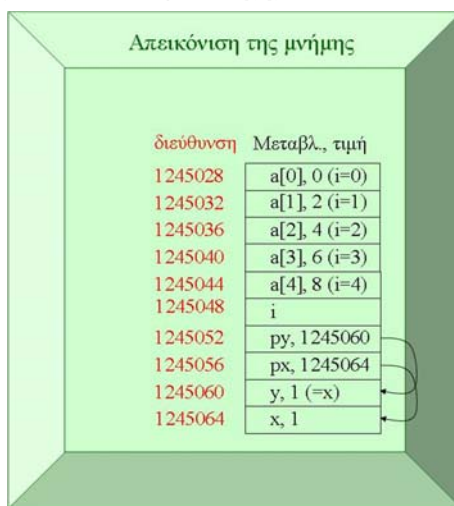
Σχήμα 6.6.η

Γραμμή 7: Το περιεχόμενο της θέσης στην οποία δείχνει ο **px**, δηλαδή το **1**, αποδίδεται στη μεταβλητή **y** (Σχήμα 6.6.θ).

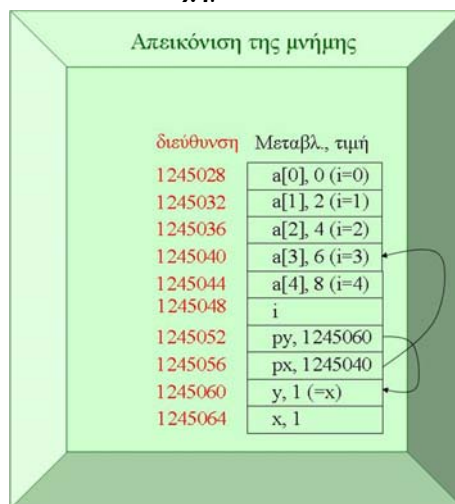
Γραμμή 8: Ανάθεση της διεύθυνσης της μεταβλητής **y** στον δείκτη **py** (Σχήμα 6.6.ι).



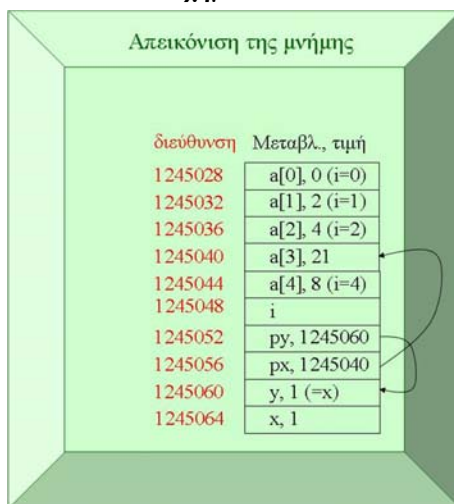
Σχήμα 6.6.θ



Σχήμα 6.6.ι



Σχήμα 6.6.ια



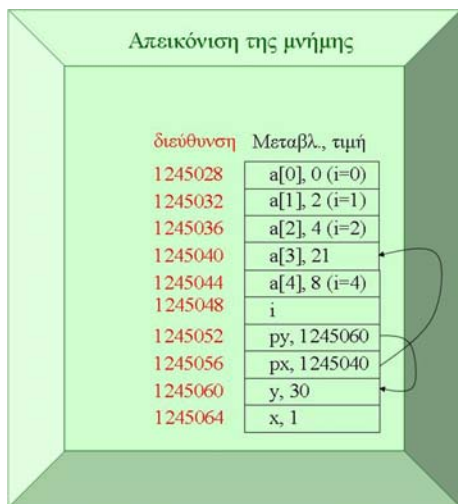
Σχήμα 6.6.ιβ

Γραμμή 9: Ο δείκτης **px** δείχνει 12 bytes (3*4) μετά τη διεύθυνση του **a[0]**, δηλαδή στη διεύθυνση του πρώτου byte του **a[3]** (Σχήμα 6.6.ια).

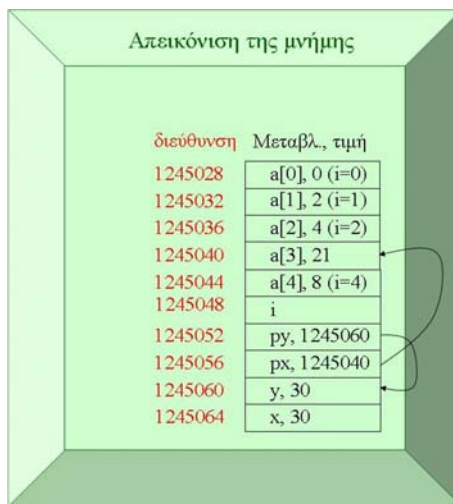
Γραμμή 10: Το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο **px** γίνεται 21 (Σχήμα 6.6.ιβ).

Γραμμή 11: Το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο **py** γίνεται ίσο με το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο **px**, αυξημένο κατά 9, δηλαδή ισούται με 30 (Σχήμα 6.6.ιγ).

Γραμμή 12: Η τιμή της μεταβλητής **x** γίνεται ίση με την τιμή της μεταβλητής **y** (Σχήμα 6.6.ιδ).



Σχήμα 6.6.ιγ



Σχήμα 6.6.ιδ

6.5 Δείκτες ως παράμετροι ή ως επιστρεφόμενοι τύποι συνάρτησης

6.5.1 Μεταβίβαση παραμέτρων – κλήση κατ' αναφορά

Στο Κεφάλαιο 4 αναλύθηκε ο τρόπος κλήσης μίας συνάρτησης και η μεταβίβαση των πραγματικών ορισμάτων στις παραμέτρους της καλούμενης συνάρτησης, σύμφωνα με τον οποίο κατά την κλήση μίας συνάρτησης οι παράμετροι αποτελούν αντίγραφο των πραγματικών ορισμάτων, καταλαμβάνοντας διαφορετικές θέσεις μνήμης. Κατά συνέπεια, η όποια επεξεργασία υφίστανται οι τυπικές παράμετροι δεν επηρεάζει τις τιμές των πραγματικών ορισμάτων. Ο παραπάνω τρόπος κλήσης ονομάζεται **κλήση κατ' αξία** (call by value).

Ωστόσο υπάρχουν περιπτώσεις, κατά τις οποίες ο συγκεκριμένος τρόπος κλήσης μίας συνάρτησης είναι ανεπαρκής και απαιτείται να δύναται η συνάρτηση να μεταβάλλει τις τιμές των πραγματικών ορισμάτων. Σε τέτοιες περιπτώσεις χρησιμοποιείται η **κλήση κατ' αναφορά** (call by reference), σύμφωνα με την οποία *δεν μεταβιβάζονται στις τυπικές παραμέτρους οι τιμές των πραγματικών ορισμάτων αλλά οι διευθύνσεις τους*. Κατά συνέπεια, η όποια επεξεργασία υποστούν οι τυπικές παράμετροι θα επηρεάσει τις τιμές των πραγματικών ορισμάτων. Η κλήση κατ' αναφορά χρησιμοποιεί τις διευθύνσεις των μεταβλητών ως πραγματικά ορίσματα και δείκτες ως τυπικές παραμέτρους.

Θα πρέπει να σημειωθεί ότι η κλήση συναρτήσεων με ορίσματα πίνακες είναι κλήση κατ' αναφορά, καθώς το όνομα ενός πίνακα αντιστοιχεί στη διεύθυνση του πρώτου byte του πρώτου στοιχείου του.

Παρατήρηση: Μπορεί να χρησιμοποιηθεί δείκτης, για να αλλαχθεί το περιεχόμενο της θέσης στην οποία δείχνει, αλλά δεν πρέπει να αλλαχθεί ο ίδιος ο δείκτης μέσα στην καλούμενη συνάρτηση. Ο λόγος είναι ότι οι πραγματικές παράμετροι, οι οποίες είναι δείκτες, αντιγράφουν μία διεύθυνση στις παραμέτρους της συνάρτησης, αλλά εάν αλλαχθεί η παράμετρος στη συνάρτηση (δηλαδή η διεύθυνση), δεν θα αλλαχθεί η πραγματική παράμετρος! Το ακόλουθο παράδειγμα αναδεικνύει το πρόβλημα.

6.5.1.1 Παράδειγμα

Να περιγραφεί αναλυτικά ο ακόλουθος κώδικας:

```
#include <stdio.h>

void print(int *ptr);

int main()
{
    int *pscore, num;
    num=32;
    pscore=&num;
    printf( "Prior to function execution,  *pscore=%d\n",*pscore );
    print(pscore);
    printf( "After function execution,  *pscore=%d\n",*pscore );

    return 0;
}

void print(int *ptr)
{
    printf( "During function execution,  *ptr=%d\n", *ptr );
    ptr++;
}
```

Αρχικά, δηλώνονται η ακέραια μεταβλητή **num**, η οποία λαμβάνει την τιμή **32**, και ο δείκτης σε ακέραιο **pscore**. Ακολούθως, στον **pscore** ανατίθεται η διεύθυνση της **num** και, στη συνέχεια, καλείται η συνάρτηση **print()** με πραγματικό όρισμα τον **pscore**. Η τυπική παράμετρος της **print()** είναι ο δείκτης σε ακέραιο **ptr**, ο οποίος λαμβάνει το περιεχόμενο του **pscore**, δηλαδή τη διεύθυνση της **num**.

Η συνάρτηση εκτυπώνει το περιεχόμενο της θέσης μνήμης, στην οποία δείχνει ο **ptr**, δηλαδή το **32**, και στη συνέχεια ο **ptr** αυξάνεται κατά ένα, δείχνοντας σε θέση μνήμης **4 bytes** κάτω από τη θέση μνήμης που έδειχνε προηγουμένως. Αυτή η μεταβολή στο περιεχόμενο του **ptr** δεν έχει νόημα, γιατί με το πέρας της συνάρτησης παύει να ισχύει ο **ptr** και δεν επιδρά στα αποτελέσματα, που παρατίθενται ακολούθως:

Prior to function execution, *pscore=32 During function execution, *ptr=32 After function execution, *pscore=32

Εικόνα 6.1 Η έξοδος του προγράμματος του παραδείγματος 6.5.1.1

6.5.1.2 Παράδειγμα

Να γίνει συγκριτική ανάλυση της λειτουργίας των ακόλουθων προγραμμάτων:

```
/* 1° πρόγραμμα */
#include <stdio.h>

void swap(int a, int b);

int main()
{
    int x=10, y=25;
    printf( "Prior to function execution:  x=%d, y=%d\n", x, y );
```

```

    swap(x,y);
    printf( "After function execution:  x=%d, y=%d\n", x, y );

    return 0;
}

void swap(int a, int b)
{
    int temp=a;
    a=b;
    b=temp;
}

```

```

/* 2ο πρόγραμμα */
#include <stdio.h>

void swap(int *pa, int *pb);

int main()
{
    int x=10, y=25;
    printf( "Prior to function execution:  x=%d, y=%d\n", x, y );
    swap(&x, &y);
    printf( "After function execution:  x=%d, y=%d\n", x, y );

    return 0;
}

void swap(int *pa, int *pb)
{
    int temp=*pa;
    *pa=*pb;
    *pb=temp;
}

```

Τα παραπάνω προγράμματα καλούν τη συνάρτηση **swap()** με σκοπό την αντιμετάθεση των περιεχομένων των μεταβλητών **x** και **y**. Στο πρώτο πρόγραμμα χρησιμοποιείται η κλήση κατ' αξία και οι τιμές των **x** και **y** αντιγράφονται στις τυπικές παραμέτρους **a** και **b**, αντίστοιχα. Αυτό που επιτυγχάνει η **swap()** είναι να αντιμεταθεθούν οι τιμές των **a** και **b**, με αποτέλεσμα μετά το πέρας της συνάρτησης **swap()** τα **a** και **b** να μην είναι πλέον ενεργά και οι τιμές των **x** και **y** να έχουν παραμείνει αμετάβλητες. Κατά συνέπεια, το πρώτο πρόγραμμα δεν εκτέλεσε επιτυχώς το έργο της αντιμετάθεσης, όπως άλλωστε προκύπτει και από τα αποτελέσματα:

<p>Prior to function execution: x=10, y=25 After function execution: x=10, y=25</p>
--

Εικόνα 6.2.α Η έξοδος του πρώτου προγράμματος του παραδείγματος 6.5.1.2

Στο δεύτερο πρόγραμμα χρησιμοποιείται η κλήση κατ' αναφορά και οι διευθύνσεις των μεταβλητών **x** και **y** αντιγράφονται στις τυπικές παραμέτρους **pa** και **pb**, αντίστοιχα, οι οποίες είναι δείκτες ακεραίων. Χρησιμοποιώντας τον τελεστή περιεχομένου (*****) και την τοπική μεταβλητή **temp**, η θέση μνήμης που αντιστοιχεί στη μεταβλητή **x**, αποκτά το περιεχόμενο της θέσης μνήμης, που αντιστοιχεί στη μεταβλητή **y** και αντίστροφα. Μετά το πέρας της συνάρτησης **swap()** παύουν να υφίστανται οι δείκτες **pa** και **pb** και το έργο της αντιμετάθεσης έχει επιτευχθεί, όπως φανερώνουν και τα αποτελέσματα:

Prior to function execution: x=10, y=25 After function execution: x=25, y=10

Εικόνα 6.2.β Η έξοδος του δεύτερου προγράμματος του παραδείγματος 6.5.1.2

6.5.1.3 Παράδειγμα

Να περιγραφεί αναλυτικά η λειτουργία του ακόλουθου προγράμματος και να δοθούν τα αποτελέσματά του.

```
#include <stdio.h>

void f1(char *pe);

int main()
{
    int a=8,*pb;
    f1("gnimmargorP larudecorP");
    pb=&a; a=14; *pb=13;
    printf( "\n\n\ta=%d\n",a );

    return 0;
}

void f1(char *pe)
{
    char *ps;
    ps=pe;
    while (*pe) pe++;
    do
    {
        pe--;
        printf( "%c",*pe );
    } while(pe>ps);
}
```

- Στη συνάρτηση **main()** ορίζονται η ακέραια μεταβλητή **a**, στην οποία δίνεται αρχική τιμή **8**, και ο δείκτης σε ακέραιο **pb**. Στη συνέχεια, καλείται η συνάρτηση **f1()** με όρισμα τη συμβολοσειρά **"gnimmargorP larudecorP"**, η οποία αποδίδεται στον δείκτη σε χαρακτήρα **pe**, δηλαδή ο **pe** θα δείχνει στη διεύθυνση του πρώτου χαρακτήρα της συμβολοσειράς, **'g'**.
- Μέσα στη συνάρτηση **f1()** ορίζεται ως τοπική μεταβλητή ο δείκτης **ps**, στον οποίο αποδίδεται το περιεχόμενο του **pe**, επομένως ο **ps** θα δείχνει στον χαρακτήρα **'g'**. Ακολουθώντας, εκτελείται ένας βρόχος **while**, ο οποίος έχει ως συνθήκη το περιεχόμενο της θέσης μνήμης να είναι διάφορο του μηδενικού χαρακτήρα. Επομένως, ο βρόχος θα εκτελεστεί τόσες φορές όσες είναι το μήκος της συμβολοσειράς (22 φορές). Σε κάθε επανάληψη του βρόχου ο δείκτης **pe** μετακινείται κατά ένα byte παρακάτω. Στο τέλος του βρόχου ο **pe** θα δείχνει στον μηδενικό χαρακτήρα τερματισμού της συμβολοσειράς **"gnimmargorP larudecorP"**.
- Ακολουθεί ένας βρόχος **do-while**, στον οποίο ο δείκτης **pe** μεταφέρεται ένα byte ψηλότερα και στη συνέχεια τυπώνεται το περιεχόμενο της διεύθυνσης στην οποία δείχνει. Ο βρόχος διαρκεί όσο ισχύει η συνθήκη **pe>ps**. Όταν η συνθήκη καταστεί ψευδής, θα έχει εκτυπωθεί η συμβολοσειρά αντεστραμμένη, δηλαδή **Procedural Programming**. Το τέλος του βρόχου οδηγεί και στον τερματισμό της κλήσης της συνάρτησης **f1()** και ο έλεγχος του προγράμματος επιστρέφει στη γραμμή

```
pb=&a;  a=14;  *pb=13;
```

Στη γραμμή αυτή ο δείκτης **pb** δείχνει στη διεύθυνση του **a**. Ακολούθως, η τιμή του **a** γίνεται **14**. Τέλος, το περιεχόμενο της διεύθυνσης, στην οποία δείχνει ο δείκτης **pb**, γίνεται **13**, δηλαδή η μεταβλητή **a** έμμεσα αποκτά την τιμή **13**. Η τιμή αυτή αποτυπώνεται στην οθόνη μέσω της τελευταίας εντολής του προγράμματος.

6.5.1.4 Παράδειγμα

Να γραφεί πρόγραμμα κωδικοποίησης δεδομένων, το οποίο θα επιτελεί τα παρακάτω:

Θα διαβάσει N τετραψήφιους αριθμούς $x = x_3x_2x_1x_0$ από το πληκτρολόγιο. Μετά από κάθε ανάγνωση αριθμού, ο x θα μετασχηματίζεται στον κωδικοποιημένο αριθμό y , ο οποίος θα εμφανίζεται στην οθόνη. Ακολουθείται η εξής μέθοδος μετασχηματισμού:

- Για κάθε αριθμό της μορφής $x_3x_2x_1x_0$, όπου ως x_k , $k=3,2,1,0$ συμβολίζονται τα τέσσερα ψηφία, υπολογίζεται ο αριθμός $y_3y_2y_1y_0$ όπου το ψηφίο y_k προκύπτει ως το υπόλοιπο της διαίρεσης του αριθμού $(x_k + 7)$ με το 10.

- Ακολούθως, αντιμετωπίζεται το πρώτο ψηφίο του αριθμού $y_3y_2y_1y_0$ με το τρίτο και το δεύτερο με το τέταρτο. Κατά συνέπεια, ο αριθμός $x_3x_2x_1x_0$ μετασχηματίζεται στον αριθμό $y_1y_0y_3y_2$.

Ο μετασχηματισμός θα υλοποιηθεί με τη συνάρτηση **int transform(int x)**, η οποία θα δέχεται τον αριθμό $x_3x_2x_1x_0$ και θα επιστρέφει τον αριθμό $y_1y_0y_3y_2$. Μέσα στη συνάρτηση **transform()** θα χρησιμοποιηθούν οι ακόλουθες συναρτήσεις:

(α) Η συνάρτηση **void give_digits(int x, int *arr)**, η οποία δέχεται ως εισόδους έναν τετραψήφιο θετικό ακέραιο $x_3x_2x_1x_0$ και έναν δείκτη που δείχνει σε πίνακα τεσσάρων θέσεων, στον οποίο αποθηκεύονται τα ψηφία του αριθμού $x_3x_2x_1x_0$.

(β) Η συνάρτηση **void swap_digits(int *arr)**, η οποία δέχεται ως είσοδο δείκτη, ο οποίος δείχνει στον πίνακα που βρίσκονται αποθηκευμένα τα ψηφία y_0, y_1, y_2, y_3 και αντιμετωπίζει τις τιμές τους σύμφωνα με την ανωτέρω μέθοδο μετασχηματισμού.

(γ) Η συνάρτηση **int get_digits(int *arr)**, η οποία δέχεται ως είσοδο δείκτη, που δείχνει σε πίνακα τεσσάρων θέσεων, στον οποίο βρίσκονται αποθηκευμένα τα τέσσερα ψηφία y_0, y_1, y_2, y_3 , και επιστρέφει τον αριθμό $y_3y_2y_1y_0$.

```
#include <stdio.h>
#define N 3

void give_digits(int x, int *arr);
int get_digits(int *arr);
int transform(int x);
void swap_digits(int *arr);

int main()
{
    int x,y,i=0;
    for (i=0;i<N;i++)
    {
        printf( "Give a number: " );
        scanf("%d",&x);
        printf( "\nCoded number=%d",transform(x) );
    }

    return 0;
```

```

}

void give_digits(int x, int *arr)
{
    int y;
    arr[3]=x/1000;
    y=x%1000;
    arr[2]=y/100;
    y=y%100;
    arr[1]=y/10;
    arr[0]=y%10;
}

int get_digits(int *arr)
{
    return(arr[0]+arr[1]*10+arr[2]*100+arr[3]*1000);
}

int transform(int x)
{
    int y,arr_x[4],arr_y[4],i,temp;
    give_digits(x,arr_x);
    for (i=0;i<4;i++)
        arr_y[i]=(arr_x[i]+7)%10;
    swap_digits(arr_y);
    return(get_digits(arr_y));
}

void swap_digits(int *arr)
{
    int temp;
    temp=arr[0];
    arr[0]=arr[2];
    arr[2]=temp;
    temp=arr[3];
    arr[3]=arr[1];
    arr[1]=temp;
}

```

<p>Give a number: 1407 Coded number=7481</p> <p>Give a number: 0000 Coded number=7777</p> <p>Give a number: 5555 Coded number=2222</p>
--

Εικόνα 6.3 Η έξοδος του προγράμματος του παραδείγματος 6.5.1.4

6.5.2 Συναρτήσεις με τύπο επιστροφής δείκτη

Έως τώρα οι δείκτες περνούσαν ως ορίσματα σε συναρτήσεις. Μπορούν, όμως, και οι συναρτήσεις να επιστρέφουν δείκτες, με την προϋπόθεση ότι ο επιστρεφόμενος δείκτης θα πρέπει να δείχνει σε δεδομένα, τα οποία θα παραμένουν ενεργά και μετά το πέρας της καλούμενης συνάρτησης (π.χ. να δείχνει σε δεδομένα της καλούσας συνάρτησης). Δεν πρέπει ποτέ να επιστρέφει δείκτης που δείχνει σε τοπική μεταβλητή της καλούμενης συνάρτησης, γιατί όταν τερματιστεί η συνάρτηση, οι τοπικές μεταβλητές εξαφανίζονται (εξαιρούνται οι static τοπικές μεταβλητές, για τις οποίες θα γίνει μνεία παρακάτω).

Μία συνάρτηση που επιστρέφει δείκτη, έχει την ακόλουθη μορφή:

<τύπος δεδομένων του επιστρεφόμενου δείκτη> *<όνομα συνάρτησης> (παράμετροι)

π.χ. στη δήλωση

```
char *incr(int x);
```

ο επιστρεφόμενος δείκτης είναι τύπου χαρακτήρα.

Για να αποφευχθούν πιθανά προβλήματα που οφείλονται στις ιδιαιτερότητες των δεικτών, προτείνεται να αποφεύγονται οι συναρτήσεις με επιστρεφόμενο δείκτη, εκτός εάν χρησιμοποιείται η λέξη κλειδί **static**.

6.5.2.1 Παράδειγμα

Θεωρούμε τον ακόλουθο κώδικα:

```
int *incr();

int main()
{
    int *pscore;
    pscore=incr();
    printf( "pscore point to %d",pscore );

    return 0;
}

int *incr()
{
    int y;
    y=10;
    return (&y);
}
```

Στον παραπάνω κώδικα υπάρχει σφάλμα, γιατί όταν τελειώνει η συνάρτηση **incr()**, η τοπική μεταβλητή **y** εξαφανίζεται και η τιμή της χάνεται. Επιστρέφοντας ο έλεγχος του προγράμματος στη **main()**, ο δείκτης **pscore** θα δείχνει στη διεύθυνση που επέστρεψε από τη συνάρτηση **incr()**, ωστόσο θα υπάρχει «σκουπίδι» (junk) σε αυτή τη διεύθυνση, εφόσον το **y** έχει παύσει να ισχύει. Μάλιστα, κατά τη μεταγλώττιση οι διάφοροι μεταγλωττιστές παρέχουν σχετικό μήνυμα προειδοποίησης (warning) για «ύποπτη μετατροπή δείκτη» (suspicious pointer conversion) ή «η συνάρτηση επιστρέφει διεύθυνση τοπικής μεταβλητής» (function returns address of local variable). Όταν αντικατασταθεί η **int y;** από **static int y;**, η μεταγλώττιση εκτελείται επιτυχώς, καθώς η στατική μεταβλητή **y** θα διατηρηθεί και μετά την έξοδο από τη συνάρτηση **incr()**.

6.6 Δείκτες και συναρτήσεις αλφαριθμητικών

Στο Κεφάλαιο 5 μελετήθηκαν ορισμένες συναρτήσεις διαχείρισης αλφαριθμητικών. Στην παρούσα ενότητα θα περιγραφούν δύο ακόμη συναρτήσεις και θα δοθούν υλοποιήσεις με χρήση δεικτών.

6.6.1 Η συνάρτηση εύρεσης χαρακτήρα σε αλφαριθμητικό

Η συνάρτηση `strchr(str1, ch)` αναζητά μέσα στο αλφαριθμητικό `str1` τον χαρακτήρα που περιέχει η μεταβλητή `ch`. Στην πρώτη εύρεση του χαρακτήρα η συνάρτηση επιστρέφει ένα δείκτη σε χαρακτήρα. Εάν δεν βρεθεί ο χαρακτήρας, επιστρέφεται η μηδενική διεύθυνση.

Η συνάρτηση `strchr()` δέχεται δύο ορίσματα: το πρώτο όρισμα είναι το όνομα του αλφαριθμητικού και το δεύτερο όρισμα είναι ο χαρακτήρας. Η συνάρτηση `strchr()` ορίζεται στο αρχείο κεφαλίδας `string.h`.

Θα πρέπει να σημειωθεί ότι ο δείκτης που επιστρέφει η συνάρτηση, ουσιαστικά αντιστοιχεί στο τμήμα του αλφαριθμητικού που αρχίζει από τον υπό εύρεση χαρακτήρα, όπως φαίνεται στο παράδειγμα που ακολουθεί.

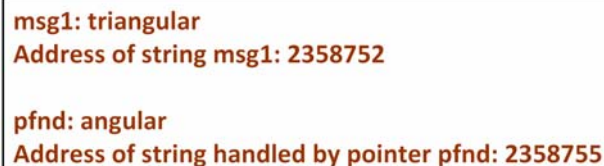
6.6.1.1 Παράδειγμα

Στο παρακάτω πρόγραμμα αναζητάται ο χαρακτήρας 'a' μέσα στο αλφαριθμητικό "triangular".

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

int main()
{
    char msg1[81]="triangular";
    char *pfnd;
    printf( "\nmsg1: %s\n",msg1 );
    printf( "Address of string msg1: %d\n",msg1 );
    pfnd = strchr(msg1, 'a' );
    printf( "\npfnd: %s\n",pfnd );
    printf( "Address of string handled by pointer pfnd: %d\n",pfnd );

    return 0;
}
```



```
msg1: triangular
Address of string msg1: 2358752

pfnd: angular
Address of string handled by pointer pfnd: 2358755
```

Εικόνα 6.4 Η έξοδος του προγράμματος του παραδείγματος 6.6.1.1

Από τα αποτελέσματα της εκτέλεσης του προγράμματος συνάγεται ότι ο δείκτης `pfnd` λαμβάνει ως τιμή το byte, στο οποίο εμφανίζεται για πρώτη φορά ο χαρακτήρας 'a' (byte 2358755). Αυτό σημαίνει ότι ο δείκτης `pfnd` μπορεί να χειριστεί ένα νέο αλφαριθμητικό, το οποίο είναι το τμήμα του αλφαριθμητικού `msg1` από το ανωτέρω byte έως το τέλος του, δηλαδή το αλφαριθμητικό "angular".

6.6.2 Η συνάρτηση εύρεσης αλφαριθμητικού σε αλφαριθμητικό

Η συνάρτηση `strstr(str1, str2)` αναζητά μέσα στο αλφαριθμητικό `str1` το αλφαριθμητικό `str2`. Στην πρώτη εύρεση του `str2` η συνάρτηση επιστρέφει έναν δείκτη σε χαρακτήρα. Εάν δε βρεθεί το `str2`, επιστρέφεται η μηδενική διεύθυνση.

Η συνάρτηση `strstr()` δέχεται δύο ορίσματα: το πρώτο όρισμα είναι το όνομα του αλφαριθμητικού, στο οποίο θα γίνει η αναζήτηση και το δεύτερο όρισμα είναι το προς εύρεση αλφαριθμητικό. Η συνάρτηση `strstr()` ορίζεται στο αρχείο κεφαλίδας `string.h`.

Όπως και στην περίπτωση της συνάρτησης `strchr()`, ο δείκτης που επιστρέφει η συνάρτηση ουσιαστικά αντιστοιχεί στο τμήμα του αλφαριθμητικού που αρχίζει από τον πρώτο χαρακτήρα του `str2` και εκτείνεται έως το τέλος του `str1`.

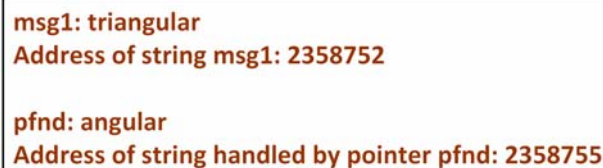
6.6.2.1 Παράδειγμα

Στο πρόγραμμα που ακολουθεί, αναζητάται το αλφαριθμητικό `'angular'` μέσα στο αλφαριθμητικό `"triangular"`.

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

int main()
{
    char msg1[81]="triangular",msg2[]={ "angular" };
    char *pfnd;
    printf( "\nmsg1: %s\n",msg1 );
    printf( "Address of string msg1: %d\n",msg1 );
    pfnd = strstr(msg1,msg2);
    printf("\n\npfnd: %s\n",pfnd);
    printf( "Address of string handled by pointer pfnd: %d\n",pfnd );

    return 0;
}
```



```
msg1: triangular
Address of string msg1: 2358752

pfnd: angular
Address of string handled by pointer pfnd: 2358755
```

Εικόνα 6.5 Η έξοδος του προγράμματος του παραδείγματος 6.6.2.1

6.6.2.2 Παράδειγμα

Στο πρόγραμμα που ακολουθεί υλοποιείται η συνάρτηση `void replace(char *pword, char *poldString, char *pnewString)`, στην οποία οι δείκτες σε χαρακτήρα `pword`, `poldString` και `pnewString` μέσω της κλήσης κατ' αναφορά χειρίζονται τα αλφαριθμητικά `word`, `fnd`, `repl`, τα οποία δίνει ο χρήστης στη συνάρτηση `main()`. Σε κάθε εμφάνιση του αλφαριθμητικού `fnd` μέσα στο αλφαριθμητικό `word`, η συνάρτηση αντικαθιστά το `fnd` με το αλφαριθμητικό `repl`. Χάριν ευκολίας, το μήκος του `fnd` είναι πάντοτε ίδιο με αυτό του `repl`. Αρχικά, πρέπει να γίνει έλεγχος κατά πόσον το `fnd` υπάρχει μέσα στο `word`.

Πέραν της ανάγνωσης των αλφαριθμητικών, η συνάρτηση `main()` επιτελεί τα ακόλουθα:

- Καλεί τη συνάρτηση `replace(,)` με ορίσματα τις διευθύνσεις των αλφαριθμητικών `word`, `fnd`, `repl`.
- Μετά το πέρας της συνάρτησης `replace()` τυπώνεται στην οθόνη το νέο περιεχόμενο του αλφαριθμητικού `word`.

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

void replace(char *pword, char *poldString, char *pnewString);

int main() {
    char word[21],fnd[11],repl[11];
    printf( "Give a word: " );          scanf( "%s",word);
    printf( "\nGive a string to find: " );  scanf( "%s",fnd );
    printf( "Give a string to replace: " );  scanf( "%s",repl );

    replace(word,fnd,repl);
    printf( "\nThe new word is: %s",word);
    return 0;
}

void replace(char *pword, char *poldString, char *pnewString)
{
    int i;
    char *temp;
    temp=strstr(pword,poldString);
    if (temp!=NULL)
    {
        do
        {
            for (i=0;i<strlen(poldString);i++)  temp[i]=pnewString[i];
            temp=temp+strlen(poldString);
            temp=strstr(temp,poldString);
        } while (temp!=NULL);
    }
    else
    {
        printf( "\n'%s' is not contained in '%s'.\n\nPress any key to
finish",poldString,pword );
        getchar();
        abort();
    }
}
```

Give a word: akakos

Give a string to find: ke

Give a string to replace: al

'ke' is not contained in 'akakos'.

Press any key to finish

(α)

Give a word: akakos

Give a string to find: ka

Give a string to replace: al

The new word is: alalos

(β)

Εικόνα 6.5 Η έξοδος του προγράμματος του παραδείγματος 6.6.2.2

6.6.3 Υλοποίηση συναρτήσεων αλφαριθμητικών με χρήση δεικτών

Στα προγράμματα που ακολουθούν δίνονται τα πρωτότυπα των συναρτήσεων:

- (α) `strlen()` (υπολογισμός του μήκους αλφαριθμητικού),
- (β) `strcpy()` (αντιγραφή ενός αλφαριθμητικού σε ένα άλλο).

(α) Θεωρώντας ότι το όρισμα που δέχεται η συνάρτηση είναι δείκτης σε χαρακτήρα, η δήλωση διαμορφώνεται ως εξής:

```
int strlen(char *p1);
```

Το σώμα της συνάρτησης με δείκτες δίνεται παρακάτω:

```
int strlen(char *p1)
{
    char *p2=p1;
    while (*p1!='\0') p1++;
    return(p1-p2);
}
```

Ο δείκτης `p1` δείχνει στη διεύθυνση του αλφαριθμητικού (στον πρώτο χαρακτήρα). Στη γραμμή κώδικα `char *p2=p1;` ο δείκτης `p2` αποκτά το περιεχόμενο του `p1`, δείχνοντας και αυτός στον πρώτο χαρακτήρα. Ακολούθως, ο βρόχος `while` εκτελείται όσο ο `p1` δεν δείχνει στον μηδενικό χαρακτήρα: σε κάθε επανάληψη ο `p1` δείχνει στον επόμενο χαρακτήρα. Όταν ο βρόχος τερματίζεται, ο `p1` δείχνει στον μηδενικό χαρακτήρα, οπότε η εντολή `p1-p2` εκτελεί αφαίρεση δεικτών και δίνει τον αριθμό των στοιχείων μεταξύ των δύο δεικτών, όπερ σημαίνει το πραγματικό μήκος του αλφαριθμητικού, δηλαδή χωρίς το μηδενικό χαρακτήρα.

(β) Ενεργώντας αντίστοιχα με το προηγούμενο σκέλος του παραδείγματος και θεωρώντας ότι τα ορίσματα που δέχεται η συνάρτηση είναι δείκτες σε χαρακτήρα, η δήλωση διαμορφώνεται ως εξής:

```
void strcpy(char *p1, char *p2);
```

Το σώμα της συνάρτησης με πίνακες και με δείκτες δίνεται παρακάτω:

```
void strcpy(char *p1, char *p2)
{
    while ((*p1=*p2)!='\0')
    {
        p1++;
        p2++;
    }
}
```

Μέσα στη συνθήκη ελέγχου της επαναληπτικής πρότασης `while` πρώτα αντιγράφεται ο χαρακτήρας, στον οποίο δείχνει ο `p2` στη θέση που δείχνει ο `p1`, και ακολούθως ελέγχεται εάν ο δείκτης `p2` έφτασε να δείχνει στον μηδενικό χαρακτήρα. Κατά συνέπεια, εφόσον ο `p2` δείξει στον μηδενικό χαρακτήρα, αυτός πρώτα θα αντιγραφεί στη θέση που δείχνει ο `p1` και μετά η συνθήκη θα καταστεί ψευδής. Με αυτόν τον τρόπο ο `p1` θα αποκτήσει στο τέλος τον μηδενικό χαρακτήρα και η αντιγραφείσα σειρά χαρακτήρων θα αποτελεί ένα ολοκληρωμένο αλφαριθμητικό.

6.7 Ορίσματα της γραμμής εντολών

Κατ' αντιστοιχία με τις υπόλοιπες συναρτήσεις της γλώσσας C, η `main()` μπορεί να δεχτεί παραμέτρους, οι οποίες επιτρέπουν να δίνεται στο καλούμενο πρόγραμμα ένα σύνολο από εισόδους, οι οποίες ονομάζονται

ορίσματα της γραμμής εντολών (command line arguments). Ο μηχανισμός μεταβίβασης ορισμάτων βασίζεται στην ακόλουθη δήλωση της **main()**:

```
void main(int argc, char *argv[])
{
    . . . . .
}
```

όπου

- **argc** (*argument count*): είναι ο αριθμός των ορισμάτων της γραμμής διαταγής, στον οποίο συμπεριλαμβάνεται το όνομα του προγράμματος. Επομένως, η τιμή του **argc** είναι πάντοτε τουλάχιστον **1**.
- **argv** (*argument vector*): είναι δείκτης σε πίνακα δεικτών, που δείχνουν στα ορίσματα της γραμμής διαταγής. Τα ορίσματα αποθηκεύονται ως αλφαριθμητικά. Ο πίνακας, στον οποίο δείχνει ο **argv**, έχει ένα επιπλέον στοιχείο, το **argv[argc]**, το οποίο έχει τιμή τη μηδενική διεύθυνση, **NULL**.

Παρατήρηση: Σε μία έκφραση δήλωσης ο τελεστής πίνακα έχει μεγαλύτερη προτεραιότητα από τον τελεστή (*). Οι δύο παρακάτω δηλώσεις βασίζονται στο γεγονός αυτό:

```
int *ar[2];
int (*ptr)[2];
```

Η πρώτη δήλωση ορίζει έναν πίνακα δύο δεικτών σε ακεραίους και στον χρόνο εκτέλεσης έχει ως αποτέλεσμα τη δέσμευση δύο θέσεων μνήμης για μελλοντική αποθήκευση δεικτών σε ακεραίους. Αντίθετα, η δεύτερη δήλωση ορίζει έναν δείκτη σε πίνακα δύο ακεραίων, τον οποίο όμως δεν δηλώνει και, κατά συνέπεια, δεν δεσμεύει τον απαιτούμενο χώρο.

6.7.1 Παράδειγμα

Στο πρόγραμμα που ακολουθεί, τυπώνονται τα ορίσματα της γραμμής διαταγής.

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

int main(int argc, char *argv[])
{
    int i;
    if (argc==1)
    {
        printf( "No arguments. Program aborted...\n" );
        exit(-1);
    }
    for (i=1;i<argc;i++)
        printf( "Argument no %d: %s",i,argv[i]\n" );

    return 0;
}
```

Έστω ότι το πρόγραμμα ονομάζεται **printArgs.c**. Όταν το καλούμε χωρίς ορίσματα, εκτελείται η πρόταση **if**, ενώ με την εντολή **printArgs first second third** το πρόγραμμα τυπώνει στην οθόνη:

```
Argument no 1: first
Argument no 2: second
Argument no 3: third
```

Το σώμα της **main()** έχει πρόσβαση στη μεταβλητή **argc**, η οποία λαμβάνει την τιμή **3**, και στον πίνακα δεικτών **argv**. Οι δείκτες που αποτελούν τα στοιχεία **argv[0]**, **argv[1]**, **argv[2]** δείχνουν στα

αλφαριθμητικά **"first"**, **"second"** και **"third"**, αντίστοιχα, ενώ το στοιχείο **argv[3]** έχει την τιμή **NULL**.

6.8 Δείκτες σε συναρτήσεις

Καθώς στη γλώσσα C οι συναρτήσεις έχουν τη δική τους διεύθυνση στη μνήμη, εκτός από τον παραδοσιακό τρόπο κλήσης με το όνομά τους μπορούν να κληθούν και μέσω δείκτη που θα «δείχνει» σε συνάρτηση (function pointer). Επιπλέον, ο χειρισμός μίας συνάρτησης μέσω δείκτη δίνει τη δυνατότητα στη συνάρτηση να αποτελέσει παράμετρο άλλης συνάρτησης. Η λειτουργία αυτή ονομάζεται callback και χρησιμοποιείται κυρίως στην κατάστρωση κώδικα για συναρτήσεις βιβλιοθήκης.

Η δήλωση δείκτη σε συνάρτηση ακολουθεί τον εξής φορμαλισμό:

<επιστρεφόμενος τύπος δεδομένων> * <όνομα δείκτη> (παράμετροι);

Έστω οι συναρτήσεις **void func1(int x, char y)** και **double *func2(int x, char y)**. Οι δείκτες συναρτήσεων δηλώνονται αντίστοιχα ως:

```
void (*pfunc1)(int, int);
double *(*pfunc2)(int, char);
```

Παρατηρήσεις:

1. Ένας δείκτης σε συνάρτηση μπορεί να δείξει σε όλες τις συναρτήσεις που συμφωνούν με τη δήλωση του δείκτη, αρκεί να οι συναρτήσεις αυτές να έχουν τον ίδιο επιστρεφόμενο τύπο δεδομένων και την ίδια λίστα ορισμάτων.
2. Συνιστάται να γίνεται αρχικοποίηση των δεικτών σε συνάρτηση με την τιμή **NULL**.
3. Όπως το όνομα ενός πίνακα ταυτίζεται με τη διεύθυνση του πρώτου byte του πρώτου στοιχείου του, έτσι και το όνομα μίας συνάρτησης – χωρίς τις παρενθέσεις – παρέχει τη διεύθυνσή της.

6.8.1 Παράδειγμα

Στο ακόλουθο πρόγραμμα παρουσιάζεται ένα παράδειγμα χρήσης των δεικτών σε συναρτήσεις:

```
#include <stdio.h>

float funcPwr(float x, int n);

int main()
{
    /* δείκτης στη συνάρτηση funcPwr() */
    float (*pfuncPwr)(float,int)=NULL;
    printf( "Address of function funcPwr=%d\n",funcPwr );
    pfuncPwr=&funcPwr;
    printf( "Pointer pfuncPwr points to address %d\n",pfuncPwr );
    printf( "\nDirect      function      call:      %.2f^%d      =
%.2f\n",2.0,6,funcPwr(2.0,6) );
    printf( "Function call via function pointer:  %.2f^%d      =
%.2f\n",2.0,6,pfuncPwr(2.0,6) );

    return 0;
}

float funcPwr(float x, int n)
{
    int i;
    float prod=1.0;
```



```

if (n==0) return 1.0;
else
{
    for (i=1;i<=n;i++)    prod=prod*x;
    return prod;
}
}

```

Address of function funcPwr=4199946
 Pointer pfuncPwr point to Address 4199946

Direct function call: $2.00^6 = 64.00$
 Function call via function pointer: $2.00^6 = 64.00$

Εικόνα 6.6 Η έξοδος του προγράμματος του παραδείγματος 6.8.1

Μελετώντας τα αποτελέσματα από την εκτέλεση του προγράμματος συμπεραίνεται ότι ο δείκτης **pfuncPwr** έχει ως περιεχόμενο τη διεύθυνση της συνάρτησης **funcPwr**. Επίσης, συνάγεται ότι τόσο η άμεση κλήση της συνάρτησης όσο και η κλήση μέσω του δείκτη **pfuncPwr** οδηγούν στη λειτουργία της συνάρτησης με τον ίδιο ακριβώς τρόπο.

Ερωτήσεις αυτοαξιολόγησης - ασκήσεις

Ερωτήσεις αυτοαξιολόγησης

Ο αναγνώστης καλείται να επιλέξει μία από τις τέσσερις απαντήσεις.

(1) Ποιος από τους ακόλουθους τρόπους αρχικοποίησης δεικτών είναι λανθασμένος;

- (α)

```
int numArray[5]={1,2,3,4,5};
int *pint, *pnum;
pint=numArray;
pnum=pint+2;
```
- (β)

```
int *pnum;
int count;
pnum=&count;
```
- (γ)

```
int numArray[5]={1,2,3,4,5};
int *pint;
pint=numArray[0];
```
- (δ)

```
int numArray[5]={1,2,3,4,5};
int *pint, *pnum;
pint=numArray;
pnum=pint;
```

(2) Ποιο είναι το αποτέλεσμα της εκτέλεσης του ακόλουθου προγράμματος;

```

#include <stdio.h>
int *func(int *p)
{
    *p=42;
    return (p) ;
};
int main()
{
    int p=13;

```

```

    int *z;
    printf( "p=%d\n",p );
    z=func(&p);
    printf( "(*z)=%d p=%d",*z,p );

    return 0;
}

```

- (α) `p=13`
`(*z)=13 p=13`
- (β) `p=13`
`(*z)=42 p=42`
- (γ) Θα εμφανιστούν σφάλματα κατά τη μεταγλώττιση.
- (δ) `p=13`
`(*z)=13 p=42`

(3) Ποιο είναι το αποτέλεσμα της εκτέλεσης του ακόλουθου προγράμματος;

```

int f(int *i)
{
    return ((*i)+1);
}
int main()
{
    int n=10;
    printf( "%d",f(&n) );

    return 0 ;
}

```

- (α) 10
- (β) Θα εμφανιστούν σφάλματα κατά τη μεταγλώττιση/αποσφαλμάτωση.
- (γ) 9
- (δ) 11

(4) Στις συναρτήσεις με τύπο επιστροφής δείκτη, ποια από τις ακόλουθες συμβάσεις για την επιστρεφόμενη τιμή είναι σωστή;

- (α) Η επιστρεφόμενη τιμή μπορεί να είναι διεύθυνση οιασδήποτε μεταβλητής της συνάρτησης.
- (β) Η C δεν υποστηρίζει συναρτήσεις με τύπο επιστροφής δείκτη.
- (γ) Η επιστρεφόμενη τιμή μπορεί να είναι διεύθυνση αθέτου μεταβλητής της συνάρτησης.
- (δ) Η επιστρεφόμενη τιμή μπορεί να είναι διεύθυνση static μεταβλητής της συνάρτησης, οιαδήποτε τύπου.

(5) Ποια από τις ακόλουθες προτάσεις είναι λανθασμένη;

- (α) Ένας δείκτης μπορεί να περιέχει μόνο τη διεύθυνση μίας θέσης μνήμης.
- (β) Ο τελεστής `*` χρησιμοποιείται για να έχουμε πρόσβαση σε μία θέση μνήμης μέσω ενός δείκτη, ο οποίος περιέχει τη διεύθυνσή της.
- (γ) Εάν εφαρμόσουμε τον τελεστή `++` σε μία μεταβλητή δείκτη, αυτή αυξάνεται κατά 1.
- (δ) Το μέγεθος μίας μεταβλητής δείκτη εξαρτάται από το σύστημα, στο οποίο εκτελείται το πρόγραμμά μας.

Ασκήσεις

Άσκηση 1

Να γραφεί πρόγραμμα, στο οποίο θα λαμβάνεται από το πληκτρολόγιο ένας ακέραιος αριθμός και θα καλείται με κλήση κατ' αναφορά η συνάρτηση `void cube(int *pnumber)`, η οποία θα υπολογίζει την τρίτη δύναμη του αναγνωσθέντος αριθμού.

Άσκηση 2

Να γραφεί πρόγραμμα, το οποίο θα επιτελεί τα ακόλουθα:

- Θα λαμβάνεται από το πληκτρολόγιο ένα αλφαριθμητικό.
- Θα καλείται με κλήση κατ' αναφορά η συνάρτηση `void rev(int *pstring)`, η οποία θα τοποθετεί τους χαρακτήρες του αλφαριθμητικού σε αντίστροφη σειρά.
- Θα καλείται με κλήση κατ' αναφορά η συνάρτηση `void conv(int *pstring)`, η οποία θα μετατρέπει τους πεζούς χαρακτήρες του αντιστραφέντος αλφαριθμητικού σε κεφαλαίους και τανάπαλιν.
- Το προκύπτον αλφαριθμητικό θα εμφανίζεται στην οθόνη.

Άσκηση 3

Να καταστρωθεί πρόγραμμα, το οποίο θα λαμβάνει από το πληκτρολόγιο δύο αλφαριθμητικά και θα τα μεταβιβάζει με κλήση κατ' αναφορά στη συνάρτηση `str_index(char *s, char *t)`. Οι δείκτες σε χαρακτήρα θα χειρίζονται τα δύο αλφαριθμητικά. Ακολούθως, μέσω του κώδικα της συνάρτησης θα υπολογίζεται η θέση της δεξιότερης εμφάνισης του αλφαριθμητικού `t` μέσα στο `s`, η οποία και θα επιστρέφει στη `main()`. Εάν το `t` δεν υπάρχει μέσα στο `s`, θα επιστρέφεται το 0. Στη `main()` θα εμφανίζεται κατάλληλο μήνυμα για τη θέση στην οποία εμφανίζεται το `t` ή για τη μη εμφάνισή του. Για την ορθή λειτουργία της συνάρτησης `str_index` απαιτείται χρήση της συνάρτησης `strstr()`, η οποία ορίζεται στο αρχείο κεφαλίδας `string.h`.

Άσκηση 4

Να περιγραφεί αναλυτικά η λειτουργία των ακόλουθων προγραμμάτων και να δοθούν τα αποτελέσματά τους.

(α) `#include <stdio.h>`

```
int func(int *i)
{
    static int j=1;
    j=j+(*i);
    return ((*i)+j);
}
int main()
{
    int n;
    for (n=0;n<8;n=n+3)
        printf( "%d\n",func(&n) );
    return 0;
}
```

(β) `#include <stdio.h>`
`#include <string.h>`

```
char *func(char *i)
{
    return(i+1);
}
int main()
{
    char *fnd,n[10]="Fourteen";
    fnd=func(&n[2]);
    printf( "%s",fnd );
    return 0;
}
```

Άσκηση 5

Να γραφεί μία συνάρτηση `int add(int *pin, int number)`, η οποία θα δέχεται ως ορίσματα: (α) τη διεύθυνση του πρώτου στοιχείου ενός μονοδιάστατου πίνακα ακεραίων και (β) το πλήθος των στοιχείων του. Η συνάρτηση θα επιστρέφει το άθροισμα των τετραγώνων των στοιχείων του πίνακα.

Άσκηση 6

Να τροποποιηθεί το πρόγραμμα του παραδείγματος 6.7.1, ώστε τα ορίσματα της γραμμής εντολών να εμφανίζονται στην οθόνη με αντρίστροφη σειρά.

Άσκηση 7

Να γραφεί πρόγραμμα, το οποίο θα επιτελεί τα ακόλουθα:

(α) Θα δημιουργείται δισδιάστατος πίνακας αριθμών κινητής υποδιαστολής `arr`, διαστάσεων 4x5.

(β) Θα καλείται η συνάρτηση `void read_data(float **parr, int m, int n)`, η οποία θα αποδίδει τιμές στον `arr` (έμμεσα) από το πληκτρολόγιο.

(γ) Θα καλούνται οι συναρτήσεις `float maxRow(float *prow, int n)`, `float minRow(float *prow, int n)`, `float meanRow(float *prow, int n)`, οι οποίες για κάθε γραμμή του πίνακα `arr` θα επιστρέφουν στη `main()` τη μέγιστη, την ελάχιστη και τη μέση τιμή των στοιχείων της γραμμής, αντίστοιχα. Θα εμφανίζονται στην οθόνη τόσο ο πίνακας `arr` όσο και οι στατιστικές τιμές για κάθε γραμμή.

Βιβλιογραφία κεφαλαίου

Θραμπουλίδης, Κ. (2002), *Διαδικαστικός Προγραμματισμός - C (Τόμος Α)*, 2^η έκδοση, Εκδόσεις Τζιόλα.

Καρολίδης, Δ. (2013), *Μαθαίνετε Εύκολα C*, αυτοέκδοση.

Deitel, H. & Deitel, P. (2014), *C Προγραμματισμός*, 7^η έκδοση, Εκδόσεις Γκιούρδα.

Kernighan, B. & Ritchie, D. (1990), *Η γλώσσα προγραμματισμού C*, Εκδόσεις Κλειδάριθμος.

Kernighan, B. & Pike, R. (1999), *The Practice of Programming*, Addison-Wesley.

Schildt, H. (1989), *Εγχειρίδιο εκμάθησης Turbo C*, Εκδόσεις Κλειδάριθμος.

Prata, S. (2014), *C Primer Plus*, 6th ed., Addison-Wesley.

Reese, R. (2013), *Understanding and Using C Pointers*, O'Reilly.

Topo, N. & Dewan, H. (2013), *Pointers in C – A Hands on Approach*, Apress.