ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Δομές Δεδομένων (Εργ.)
Ακ. Έτος 2017-18
Διδάσκων: Ευάγγελος Σπύρου

Επανάληψη Γ' μέρος

1. Στόχος του εργαστηρίου

Στόχος του τρίτου μέρους της επανάληψης είναι η υπενθύμιση βασικών γνώσεων στη γλώσσα C. Όλες οι δομές δεδομένων που θα διδαχθούν στο εργαστήριο, θα αναπτυχθούν σε C. Για το λόγο αυτό, απαιτείται εξοικείωση με τις βασικές έννοιες προγραμματισμού που έχουν ήδη διδαχθεί.

2. Δείκτες

Οι δείκτες είναι μια από τις σημαντικότερες, αλλά ίσως και η πιο δυσνόητη ενότητα της γλώσσας C. Όπως ίσως γνωρίζουμε, η μνήμη RAM ενός υπολογιστή αποτελείται από εκατομμύρια θέσεις αποθήκευσης δεδομένων. Κάθε θέση μνήμης μπορεί να αποθηκεύσει 8 bits δεδομένων. Προσδιορίζεται από μοναδικό αριθμό που ονομάζεται διεύθυνση μνήμης.

Όταν δηλώνεται μια μεταβλητή, ο μεταγλωττιστής δεσμεύει τις απαραίτητες συνεχόμενες θέσεις στη μνήμη για να αποθηκεύσει την τιμή της. Όταν μια μεταβλητή καταλαμβάνει πολλές θέσεις μνήμης, τότε διεύθυνση μιας μεταβλητής θεωρείται ότι είναι η διεύθυνση της πρώτης θέσης μνήμης. Για παράδειγμα, όταν κάνουμε τη δήλωση `int a = 10;` ο μεταγλωττιστής βρίσκει τέσσερις ελεύθερες θέσεις μνήμης και τις δεσμεύει. Αν για παράδειγμα η διεύθυνση της μεταβλητής `a` αρχίζει στη θέση `5000`, τότε η τιμή της θα αποθηκευτεί στις θέσεις από `5000` εώς και `5003`. Ο μεταγλωττιστής συσχετίζει το όνομα μιας μεταβλητής με τη διεύθυνσή της. Όταν η μεταβλητή χρησιμοποιείται στο πρόγραμμα, ο μεταγλωττιστής προσπελαύνει τη διεύθυνσή της. Για παράδειγμα, με την εντολή `a = 80;` ο μεταγλωττιστής γνωρίζει ότι η διεύθυνση της `a` είναι η 5000 και θέτει το περιεχόμενό της ίσο με `80`.

3. Δήλωση Δεικτών

Ένας δείκτης είναι μια μεταβλητή στην οποία μπορεί να αποθηκευτεί μια διεύθυνση μνήμης και δηλώνεται ως εξής:

```
τύπος_δεδομένων *όνομα_δείκτη;
```

Για παράδειγμα με τη δήλωση `int *ptr;` η μεταβλητή `ptr` είναι ένας δείκτης σε τύπο `int`. Ο τύπος της έκφρασης `*ptr` είναι `int`. Στον `ptr` μπορεί να εκχωρηθεί η διεύθυνση κάποιας ακέραιας μεταβλητής. Πριν από το όνομα, πρέπει να προηγείται ο χαρακτήρας `*`. Παρομοίως με τη δήλωση `double *ptr;` μπορεί να εκχωρηθεί στο δείκτη `ptr` η διεύθυνση κάποιας πραγματικής μεταβλητής τύπου `double`.

Οι δείκτες μπορούν να δηλώνονται και μαζί με άλλες μεταβλητές. Π.χ.:

```
int *ptr, i, j, k;
```

Σημειώστε ότι το * μπορεί να τοποθετηθεί δίπλα στον τύπο (π.χ. int* ptr;), ωστόσο σε περίπτωση που επιθυμούμε να δηλώσουμε περισσότερους δείκτες, μπορεί να οδηγήσει σε σύγχυση. Για παράδειγμα με τη δήλωση:

```
int* p1, p2;
```

Ο p1 δηλώνεται σαν δείκτης, ο p2 όχι. Όπως και με τις απλές μεταβλητές, όταν δηλώνεται ένας δείκτης, ο μεταγλωττιστής δεσμεύει τις απαραίτητες θέσεις μνήμης για να αποθηκεύσει την τιμή του. Θυμηθείτε, ότι με τον τελεστή sizeof μπορούμε να δούμε πόσες οκτάδες μνήμης δεσμεύονται από έναν δείκτη:

```
#include<stdio.h>
int main(void)
{
     int *ptr;
     printf("Bytes: %lu\n", sizeof(ptr));

     return 0;
}
```

4. Απόδοση τιμής σε δείκτη

Αφού δηλώσουμε έναν δείκτη, μπορούμε να του εκχωρήσουμε τη διεύθυνση μνήμης κάποιας μεταβλητής. Για να βρούμε τη διεύθυνση μιας μεταβλητής, χρησιμοποιούμε τον τελεστή & (address operator), πριν από το όνομά της. Για παράδειγμα:

```
#include <stdio.h>
int main(void)
{
     int *ptr, a;
     ptr = &a;
     printf("Address= %p\n", ptr);

     return 0;
}
```

Εδώ, με την εντολή ptr = &a; η τιμή του δείκτη ptr γίνεται ίση με τη διεύθυνση της μεταβλητής a και συνηθίζεται να λέμε ότι ο ptr "δείχνει" στο a. Συνήθως, για την εμφάνιση μιας διεύθυνσης μνήμης χρησιμοποιείται το %p το οποίο και την εμφανίζει σε δεξαεξαδική μορφή. Η απευθείας ανάθεση μιας ακέραιας τιμής σε έναν δείκτη είναι πολύ πιθανό να προκαλέσει σφάλμα μεταγλώττισης:

```
int *ptr;
```

```
ptr = 10000;
```

Δοκιμάστε το για να δείτε πώς συμπεριφέρεται ο μεταγλωττιστής που χρησιμοποιείτε. Η ανάθεση τιμής σε έναν δείκτη μπορεί να γίνει ταυτόχρονα με τη δήλωσή του, αρκεί να έχει προηγηθεί η δήλωση της μεταβλητής στην οποία θα δείχνει ο δείκτης:

```
int a, b, *ptr = &a;
```

Ο τελεστής `&` δεν μπορεί να εφαρμοστεί σε σταθερές και παραστάσεις, αλλά μόνο σε στοιχεία που είναι στη μνήμη, όπως μεταβλητές, στοιχεία πίνακα και συναρτήσεις.

5. Μηδενικοί δείκτες

Υπενθυμίζεται ότι η αρχική τιμή μιας μεταβλητής είναι το περιεχόμενο της διεύθυνσης μνήμης της. Επομένως, μια μεταβλητή-δείκτης αρχικοποιείται με μια τυχαία τιμή που αντιστοιχεί σε κάποια διεύθυνση μνήμης. Όταν θέλουμε να δηλώσουμε ρητά ότι ένας δείκτης δεν δείχνει πουθενά, του αναθέτουμε μια ειδική τιμή και πλέον ο δείκτης λέγεται "μηδενικός δείκτης" (null pointer). Αυτή η τιμή είναι το `0` και αντιπροσωπεύεται από τη σταθερά `NULL` (δηλώνεται στο `stdio.h`). Δείτε τι θα τυπώσει το παρακάτω παράδειγμα:

```
#include <stdio.h>
int main(void)
{
     int *ptr;

     printf("Addr= %p\n", ptr);
     ptr = NULL;
     printf("Addr= %p\n", ptr);

     return 0;
}
```

Η τιμή ενός δείκτη μπορεί να συγκριθεί με την τιμή `NULL` ως εξής:

```
if (ptr != NULL) //ισοδύναμο με if(ptr)
if (ptr == NULL) //ισοδύναμο με if(!ptr)
```

6. Χρήση Δείκτη

Για να προσπελάσουμε το περιεχόμενο κάποιας διεύθυνσης μνήμης με χρήση δείκτη χρησιμοποιούμε τον τελεστή `*` (dereference/indirection operator) πριν από το όνομα του δείκτη. Προσοχή, δεν έχει σχέση με πολλαπλασιασμό, αν και είναι το ίδιο σύμβολο. Δείτε το παρακάτω παράδειγμα:

```
#include <stdio.h>
int main(void)
{
```

```
    int *ptr, a;

    a = 10;
    ptr = &a;
    printf("Val= %d\n", *ptr);

    return 0;
}
```

Η έκφραση `*ptr` ισοδυναμεί με το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο `ptr`. Αφού ο `ptr` δείνχει στη διεύθυνση της μεταβλητής `a`, το `*ptr` θα είναι ίσο με την τιμή του `a`. Άρα το πρόγραμμα θα εμφανίσει `10`.

Δείτε το παρακάτω πρόγραμμα που θα εμφανίσει μήνυμα λάθους, καθώς ο `ptr` δεν δείχνει στη διεύθυνση κάποιας μεταβλητής, προτού χρησιμοποιηθεί στην εντολή `a = *ptr;`

```
#include <stdio.h>
int main(void)
{
    int *ptr, a;
    a = *ptr;
    printf("Val = %d\n", a);

    return 0;
}
```

Το επόμενο πρόγραμμα ωστόσο εκτελείται κανονικά, καθώς ο `ptr` δείχνει στη διεύθυνση κάποιας μεταβλητής προτού χρησιμοποιηθεί στην εντολή `i = *ptr;`

```
#include <stdio.h>
int main(void)
{
    int *ptr, i, j;
    j = 20;
    ptr = &j;

    i = *ptr;
    printf("Val = %d\n", i);

    return 0;
}
```

Προσοχή: οι τελεστές `*` και `&` αλληλοαναιρούνται, όταν χρησιμοποιούνται μαζί. Δείτε τι θα εμφανίσει το παρακάτω πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
     int *ptr, i;
     ptr = &i;
     printf("%p %p %p\n", &i, *&ptr, &*ptr);

     return 0;
}
```

Άσκηση 3.1
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
     int *ptr, i = 10;
     ptr = &i;
     i += 20;
     printf("%d\n", *ptr);

     return 0;
}
```

Άσκηση 3.2
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο ακεραίους, να δηλώνει δύο δείκτες προς αυτούς και να εμφανίζει τις διευθύνσεις μνήμης των ακεραίων μεταβλητών, τα περιεχόμενα των δεικτών και τις διευθύνσεις μνήμης των δεικτών.

```
#include <stdio.h>
int main(void)
{
     int *ptr1, *ptr2, i, j;

     printf("Enter numbers: ");
     scanf("%d%d", &i, &j);

     ptr1 = &i;
     ptr2 = &j;

     printf("Num1 address= %p\n", ptr1);
     printf("Num2 address= %p\n", ptr2);
```

```
     printf("Ptr1 content= %d\n", *ptr1);
     printf("Ptr2 content= %d\n", *ptr2);

     printf("Ptr1 address= %p\n", &ptr1);
     printf("Ptr2 address= %p\n", &ptr2);

     return 0;
}
```

Άσκηση 3.3

Ο σκοπός του παρακάτω προγράμματος είναι να διαβάσει έναν πραγματικό αριθμό με χρήση δείκτη και να τον εμφανίσει. Υπάρχει λάθος στον κώδικα;

```
#include <stdio.h>
int main(void)
{
     double *ptr,  i;
     scanf("%lf", ptr);
     printf("Val= %f\n", *ptr);

     return 0;
}
```

Άσκηση 3.4

Να γραφεί ένα πρόγραμμα το οποίο με χρήση δείκτη να διαβάζει μια πραγματική τιμή και να εμφανίζει την απόλυτη τιμή της.

```
#include <stdio.h>
int main(void)
{
     double *p, val;
     p = &val;
     printf("Enter number: ");
     scanf("%lf", p);

     if (*p >= 0)
          printf("%f\n", *p);
     else
          printf("%f\n", -*p);

     return 0;
}
```

Άσκηση 3.5

Να γραφεί ένα πρόγραμμα το οποίο με χρήση δεικτών να διαβάζει δύο πραγματικές τιμές και να τις αντιμεταθέτει. Στη συνέχεια να εμφανίζει τη μεγαλύτερη τιμή.

Άσκηση 3.6
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
     int *ptr1, i = 10;
     double *ptr2, j = 1.234;

     ptr1 = &i;
     ptr2 = &j;

     *ptr1 = *ptr2;
     printf("%d  %lu  %lu  %lu\n",  i,  sizeof(ptr1),  sizeof(ptr2),
sizeof(*ptr2));

     return 0;
}
```

Άσκηση 3.7
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
     int *ptr1, *ptr2, i = 10, j = 20;

     ptr1 = &i;
     ptr2 = &j;

     ptr2 = ptr1;
     *ptr1 = *ptr1 + *ptr2;
     *ptr2 *= 2;

     printf("%d\n", *ptr1+*ptr2);
     return 0;
}
```

7. Αριθμητική Δεικτών

Η αριθμητική δεικτών αφορά την εκτέλεση αριθμητικών πράξεων με δείκτες. Σύμφωνα με το πρότυπο, η αριθμητική δεικτών παράγει αξιόπιστα αποτελέσματα όταν εφαρμόζεται σε στοιχεία του ίδιου πίνακα, αλλιώς το αποτέλεσμα είναι απροσδιόριστο. Οι επιτρεπτές πράξεις είναι η πρόσθεση ακέραιου σε δείκτη, η αφαίρεση ακεραίου από δείκτη και η αφαίρεση δύο δεικτών.

Η πρόσθεση ενός θετικού ακεραίου n σε έναν δείκτη σε μια εντολή όπως:

```
ptr = ptr + n;
```

αυξάνει την τιμή του δείκτη κατά `n*`*μέγεθος* του τύπου στον οποίο δείχνει και τον κάνει να δείχνει στη διεύθυνση του `n`-οστού στοιχείου μετά από αυτό που έδειχνε. Αν το αποτέλεσμα της πράξης είναι εκτός των ορίων του πίνακα, το αποτέλεσμα είναι απροσδιόριστο.

Δείτε το παρακάτω πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
     int *ptr, arr[] = {10, 20, 30};

     ptr = &arr[0];
     printf("Addr: %p\n", ptr);
     ptr = ptr+2;
     printf("Addr: %p Val: %d\n", ptr, *ptr);

     return 0;
}
```

Παρόμοια με την πρόσθεση, η αφαίρεση ενός θετικού ακεραίου από έναν δείκτη σε μια εντολή εκχώρησης όπως:

```
ptr = ptr – n;
```

μειώνει την τιμή του κατά `n *` *μέγεθος* του τύπου του και θα δείχνει στη διεύθυνση του `n`-οστού στοιχείου πριν από αυτό που έδειχνε. Δοκιμάστε να τροποποιήσετε κατάλληλα το προηγούμενο πρόγραμμα.

Σημειώστε ότι αν ο δείκτης δεν δείχνει σε στοιχείο πίνακα, η εφαρμογή των τελεστών `++`/`--` είναι έγκυρη και προκαλεί την αύξηση/μείωση του δείκτη κατά το μέγεθος του τύπου στον οποίο δείχνει. Δείτε το παρακάτω παράδειγμα:

```
#include <stdio.h>
int main(void)
{
     double *ptr, i;
```

```
    ptr = &i;
    ptr++;
    printf("Addr:%d\n", ptr);
    ptr--;
    printf("Addr:%d\n", ptr);

    return 0;
}
```

Το πρόγραμμα εμφανίζει δύο διευθύνσεις με την πρώτη να είναι μεγαλύτερη κατά `8` θέσεις από τη δεύτερη. Η συνδυαστική χρήση των τελεστών `++`/`--` με τον τελεστή `*` είναι πολύ συνηθισμένη στη διαχείριση πινάκων με χρήση δείκτη. Το αποτέλεσμα της έκφρασης εξαρτάται από τη θέση των τελεστών με βάση τον πίνακα προτεραιοτήτων. Υπάρχουν οι εξής περιπτώσεις:

`i = (*ptr)++;` πρώτα εκχωρείται η τιμή του `*ptr` στο `i` και έπειτα αυξάνεται η τιμή του `*ptr` κατά ένα
`i = *ptr++;` πρώτα εκχωρείται η τιμή του `*ptr` στο `i` και έπειτα αυξάνεται η τιμή του `ptr` ανάλογα με τον τύπο του
`i = ++*ptr;` πρώτα αυξάνεται κατά ένα η τιμή του `*ptr` και μετά αυτή εκχωρείται στο `i`
`i = *++ptr;` πρώτα αυξάνεται η τιμή του `ptr` και μετά αυτή εκχωρείται στο `i` η τιμή του `*ptr`

Το αποτέλεσμα της αφαίρεσης δεικτών είναι ο αριθμός των στοιχείων που μεσολαβούν μεταξύ τους. Οι δείκτες πρέπει να δείχνουν σε στοιχεία του ίδιου πίνακα ή στην αμέσως επόμενη θέση από το τέλος του πίνακα. Αν η τιμή του δείκτη που αφαιρείται είναι μεγαλύτερη, τότε το αποτέλεσμα θα είναι το ίδιο, απλά με αρνητικό πρόσημο. Για παράδειγμα, αν ο `p1` δείχνει στο δεύτερο στοιχείο και ο `p2` στο πέμπτο στοιχείο του ίδιου πίνακα, το αποτέλεσμα της `p2-p1` είναι `3`, ενώ της `p1-p2` είναι `-3`.

Το αποτέλεσμα της σύγκρισης δύο δεικτών με τους τελεστές `==` και `!=` είναι αξιόπιστο. Με τους τελεστές `<`, `<=`, `>` και `>=` το αποτέλεσμα θεωρείται αξιόπιστο αν οι δείκτες δείχνουν στο ίδιο αντικείμενο, π.χ. πίνακας ή δομή, αλλιώς είναι απροσδιόριστο. Για παράδειγμα, η τιμή του `p2 > p1` είναι `1`.

Άσκηση 3.8
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
    int *ptr, i = 0;

    for(ptr = &i; *ptr < 5; i++)
    {
```

```
            (*ptr)++;
            ++*ptr;
            printf("%d ", i);
      }
      return 0;
}
```

Άσκηση 3.9

Να γραφεί ένα πρόγραμμα το οποίο με χρήση δεικτών να διαβάζει τους βαθμού ενός φοιτητή σε τρεις εργασίες. Αν και οι τρεις βαθμοί είναι μεγαλύτεροι ή ίσοι του `5`, το πρόγραμμα να τους εμφανίζει με αύξουσα σειρά, διαφορετικά να εμφανίζει το μέσο όρο τους.

9. Δείκτες και Πίνακες

Όπως έχουμε αναφέρει, τα στοιχεία ενός πίνακα αποθηκεύονται σε διαδοχικές θέσεις μνήμης, με το πρώτο στοιχείο στη χαμηλότερη διεύθυνση. Τα επόμενα στοιχεία αποθηκεύονται στις υψηλότερες διευθύνσεις σύμφωνα με τον τύπο δεδομένων του πίνακα. Για παράδειγμα σε έναν πίνακα χαρακτήρων, οι διευθύνσεις των στοιχείων απέχουν μεταξύ τους μία οκτάδα ενώ σε έναν πίνακα ακεραίων τέσσερις(;) οκτάδες. Σημειώνουμε ότι η στενή σχέση πίνακα και δείκτη βασίζεται στο ότι το όνομα ενός πίνακα μπορεί να χρησιμοποιηθεί ως δείκτης στο πρώτο στοιχείο του πίνακα.

Έστω η δήλωση `int arr[3];` Αν θεωρήσουμε ότι η διεύθυνση του πρώτου στοιχείου είναι η θέση `100`, η τιμή του πρώτου στοιχείου αποθηκεύεται στις θέσεις `100–103`, η τιμή του δεύτερου στοιχείου στις θέσεις `104–107` και του τρίτου στις θέσεις `108–111`. Παρόμοια, η έκφραση `arr+1` είναι ένας δείκτης που δείχνει στο δεύτερο στοιχείο του πίνακα κ.ο.κ. Στη γενική περίπτωση, οι ακόλουθες εκφράσεις είναι ισοδύναμες:

```
arr = &arr[0]
arr + 1 == &arr[1]
arr + 2 == &arr[2]
...
arr + n == &arr[n]
```

Δηλαδή, το παρακάτω πρόγραμμα εμφανίζει την ίδια τιμή.

```
#include <stdio.h>
int main(void)
{
      int *ptr, arr[5];

      ptr = arr;
      printf("%p %p %p %p\n", ptr, &arr[0], arr, &arr);
      return 0;
}
```

Σημειώστε ότι η τελευταία έκφραση `&arr` αν και εμφανίζει την ίδια τιμή με τις υπόλοιπες, είναι διαφορετική. Ισοδυναμεί με έναν δείκτη σε όλον τον πίνακα, ενώ οι υπόλοιπες με έναν δείκτη στο πρώτο στοιχείο του. Αφού το όνομα ενός πίνακα μπορεί να χρησιμοποιηθεί σαν δείκτης στο πρώτο στοιχείο του, το περιεχόμενό του θα είναι ίσο με την τιμή του πρώτου στοιχείου. Δηλαδή, η τιμή του `*arr` είναι ίση με `arr[0]`. Παρόμοια, αφού το `arr+1` είναι ένας δείκτης στο δεύτερο στοιχείο του πίνακα, το `*(arr+1)` είναι ίσο με `arr[1]` κ.ο.κ. Στη γενική περίπτωση, οι ακόλουθες εκφράσεις είναι ισοδύναμες:

```
*arr = arr[0]
*(arr+1) == arr[1]
*(arr+2) == arr[2]
...
*(arr+n) == arr[n]
```

Οι παρενθέσεις είναι απαραίτητες γιατί ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`. Δηλαδή οι εκφράσεις `*(arr+n)` και `*arr+n` εκτελούνται με διαφορετικό τρόπο. Δείτε για παράδειγμα το παρακάτω πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
     int *ptr, arr[] = {10, 20, 30, 40, 50};
     ptr = arr;
     printf("Val1= %d Val2 = %d\n", *ptr+2, *(ptr+2));

     return 0;
}
```

Με την εντολή `ptr = arr;` η τιμή του `ptr` γίνεται ίση με τη διεύθυνση του `arr[0]`, άρα η τιμή του `*ptr` είναι ίση με `arr[0]` δηλαδή `10`. Αφού ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον τελεστή `+`, το αποτέλεσμα της `*ptr+2` είναι `10+2=12`. Αντίθετα, η έκφραση `*(ptr+2)` ισοδυναμεί με το περιεχόμενο της διεύθυνσης που δείχνει ο `ptr`, αυξημένης κατά δύο θέσεις ακεραίων. Δηλαδή το `*(ptr+2)` είναι ισοδύναμο με το `arr[2]`. Άρα το πρόγραμμα θα εμφανίσει `Val1 = 12 Val2 = 30`.

Προσοχή: δεν γράφουμε `ptr = &arr;` γιατί το `ptr` είναι δείκτης σε ακέραιο, ενώ το `&arr` αντιστοιχεί σε δείκτη σε πίνακα. Δείτε στο επόμενο πρόγραμμα πώς μπορούμε να εμφανίσουμε τις διευθύνσεις μνήμης και τις τιμές των στοιχείων του πίνακα arr με δύο διαφορετικούς τρόπους:

```
#include <stdio.h>
int main(void)
{
     int i, arr[] = {10, 20, 30, 40, 50};
```

```
     printf("using array notation:\n");
     for(i = 0; i < 5; i++)
          printf("Addr = %p Val = %d\n", &arr[i], arr[i]);

     printf("using pointer notation:\n");
     for(i = 0; i < 5; i++)
          printf("Addr = %p Val = %d\n", arr+i, *(arr+i));

     return 0;
}
```

Με τον πρώτο τρόπο χρησιμοποιούμε τη θέση του στοιχείου στον πίνακα, με τον δεύτερο το όνομα του πίνακα σαν δείκτη. Ο πρώτος τρόπος οδηγεί σε πιο ευανάγνωστο κώδικα.

Άσκηση 3.10
Ποια είναι η έξοδος του παρακάτω προγράμματος:

```
#include<stdio.h>
int main(void)
{
     int i = 10, *p = &i;

     p[0] = 50;
     printf("%d\n", i);

     return 0;
}
```

Άσκηση 3.11
Ποια είναι η έξοδος του παρακάτω προγράμματος:

```
#include<stdio.h>
int main(void)
{
     int i, *ptr1, *ptr2, arr[] = {10, 20, 30, 40, 50, 60, 70};

     ptr1 = &arr[2];
     ptr2 = &arr[4];

     for(i = ptr2-ptr1; i < 5; i+=2)
          printf("%d ", ptr1[i]);

     return 0;
}
```

Άσκηση 3.12
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς 10 φοιτητών, να τους αποθηκεύει σε έναν πίνακα και να εμφανίζει τη μικρότερη και τη μεγαλύτερη τιμή, καθώς και τις θέσεις της πρώτης εμφάνισης τους στον πίνακα. Το πρόγραμμα να ελέγχει εάν ο εισαγόμενος βαθμός ανήκει στο [0, 10]. Η διαχείριση του πίνακα να γίνει με σημειογραφία δείκτη.

10. Πίνακας δεικτών
Ένας πίνακας δεικτών είναι ένα πίνακας που τα στοιχεία του είναι δείκτες στον ίδιο τύπο δεδομένων. Για να δηλώσουμε έναν πίνακα δεικτών, χρησιμοποιούμε το * πριν από το όνομα του πίνακα. Για παράδειγμα με τη δήλωση:

```
int *p[3];
```

ο πίνακας p περιέχει τρία στοιχεία, όπου το καθένα από αυτά είναι δείκτης σε μια ακέραια μεταβλητή. Τα στοιχεία ενός πίνακα δεικτών τα χειριζόμαστε με τον ίδιο τρόπο όπως και τους απλούς δείκτες. Π.χ.:

```
#include<stdio.h>
int main(void)
{
     int *p[3], i = 100, j = 200, k = 300;

     p[0] = &i;
     p[1] = &j;
     p[2] = &k;
     printf("%d %d %d\n", *p[0], *p[1], *p[2]);

     return 0;
}
```

Άσκηση 3.13
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
     int *p[3], i, num;

     for(i = 0; i < 3; i++)
     {
          printf("Enter number: ");
          scanf("%d", &num);
          p[i] = &num;
```

```
    }
    for(i = 0; i < 3; i++)
        printf("Num: %d\n", *p[i]);

    return 0;
}
```

Άσκηση 3.14
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
    int *p[3], i, arr[4] = {10, 20, 30, 40};

    for(i = 0; i < 3; i++)
    {
        p[i] = &arr[i] + 1;
        printf("%d ", *p[i]);
    }
    return 0;
}
```

11. Δείκτης σε δείκτη
Όταν δηλώνεται ένας δείκτης, ο μεταγλωττιστής δεσμεύει τις απαραίτητες θέσεις μνήμης, όπως κάνει για οποιαδήποτε μεταβλητή, για να αποθηκεύσει την τιμή του. Επομένως, αφού δεσμευτεί μια διεύθυνση μνήμης για έναν δείκτη, μπορούμε να δηλώσουμε έναν άλλο δείκτη που δείχνει προς την ίδια διεύθυνση. Για να δηλώσουμε μια μεταβλητή που είναι δείκτης σε δείκτη, χρησιμοποιούμε δύο φορές το `*`. Π.χ. με τη δήλωση:

```
int **p;
```

η μεταβλητή `pp` δηλώνεται σαν δείκτης προς κάποιον άλλον δείκτη, ο οποίος με τη σειρά του δείχνει προς μια ακέραια μεταβλητή. Αν έχουμε δηλώσει ένα δείκτη σε δείκτη, τότε με ένα `*` αποκτούμε πρόσβαση στη διεύθυνση του δεύτερου δείκτη, ενώ με διπλό `**` στην τιμή της μεταβλητής που δείχνει ο δεύτερος δείκτης. Δείτε π.χ. το ακόλουθο πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
    int *p, **pp, i = 20;

    p = &i;
    pp = &p;
```

```
     printf("%d\n", **pp);

     return 0;
}
```

Η συνηθέστερη χρήση ενός δείκτη σε δείκτη είναι όταν επιθυμούμε μια συνάρτηση να μπορεί να μεταβάλει την τιμή κάποιου δείκτη. Γενικά επιτρέπονται πολλαπλές δηλώσεις δεικτών προς δείκτες, αλλά συνήθως δεν γίνεται υπέρβαση των δύο επιπέδων.

Άσκηση 3.15
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
     int *p, **pp, i = 10, j = 20;

     p = &i;
     pp = &p;
     **pp += 100;

     p = &j;
     **pp += 100;
     printf("%d\n", i+j);

     return 0;
}
```

Άσκηση 3.16
Ποιες τιμές αποκτούν τα στοιχεία του πίνακα `a` στο παρακάτω πρόγραμμα;

```
#include <stdio.h>
int main(void)
{
     int k = 0, b = 1, c = 2, d = 3, m, a[3];
     int *p[] = {&k, &b, &c, &d};

     for(m = 0; m < 3; m++)
          a[*p[m]] = **(p+m+1);

     return 0;
}
```

13. Συναρτήσεις

Στην υποενότητα αυτή θα θυμηθούμε τα βασικά για τη δήλωση, την υλοποίηση και τη χρήση συναρτήσεων. Θυμίζουμε ότι μια συνάρτηση είναι ένα ανεξάρτητο τμήμα κώδικα που εκτελεί μια ορισμένη εργασία και προαιρετικά επιστρέφει μια τιμή Η συγγραφή προγραμμάτων με χρήση συναρτήσεων που εκτελούν ανεξάρτητες εργασίες είναι η βάση του δομημένου προγραμματισμού.

Η δήλωση (ή πρωτότυπο) μιας συνάρτησης καθορίζει το όνομα της συνάρτησης, τον τύπο επιστροφής και μια λίστα παραμέτρων. Η γενική μορφή δήλωσης μιας συνάρτησης είναι:

```
τύπος_επιστροφής όνομα_συνάρτησης(λίστα_παραμέτρων)
```

Καλό είναι να χρησιμοποιείτε περιγραφικά ονόματα για τις συναρτήσεις σας, βοηθώντας έτσι τον αναγνώστη του κώδικά σας να κατανοεί ευκολότερα τον σκοπό τους. Συνήθως, οι δηλώσεις των συναρτήσεων περιέχονται σε ξεχωριστό αρχείο. Για παράδειγμα, οι συναρτήσεις βιβλιοθήκης δηλώνονται σε διάφορα αρχεία επικεφαλίδας. Για κάθε συνάρτηση βιβλιοθήκης που χρησιμοποιούμε στα προγράμματά μας, συμπεριλαμβάνουμε με την οδηγία `#include` το αρχείο που περιέχει τη δήλωσή της.

Μια συνάρτηση μπορεί να επιστρέψει το πολύ μια τιμή. Ο τύπος_επιστροφής καθορίζει τον τύπο της τιμής επιστροφής. Μπορεί να είναι οποιοσδήποτε τύπος δεδομένων (`char`, `int`, `float` κτλ.) ή δείκτης σε κάποιον τύπο. Προσοχή: μια συνάρτηση δεν μπορεί να επιστρέψει κάποιον πίνακα, αλλά μπορεί να επιστρέψει δείκτη σε πίνακα.

Συνήθως, αν ο τύπος επιστροφής της συνάρτησης παραλειφθεί, θεωρείται ότι η συνάρτηση επιστρέφει `int`. Για να δηλώσουμε ότι μια συνάρτηση δεν επιστρέφει κάποια τιμή, χρησιμοποιούμε τη δεσμευμένη λέξη `void`.

Μια συνάρτηση μπορεί να δέχεται μια λίστα παραμέτρων, οι οποίες χωρίζονται μεταξύ τους με κόμμα. Κάθε παράμετρος χαρακτηρίζεται από τον τύπο της. Η παράμετρος είναι ουσιαστικά μια μεταβλητή της συνάρτησης, η οποία θα αρχικοποιηθεί με μία τιμή, όταν γίνει η κλήση της. Όταν μια συνάρτηση δεν δέχεται παραμέτρους, η λίστα παραμέτρων μπορεί να δηλωθεί ως `void`, προκειμένου να υπάρχει πλήρης συμφωνία με το πρότυπο.

Δείτε τα παρακάτω παραδείγματα:

```
void show(char ch);

double show(int a, float b);

int* show(int *p, double a);
```

Τα ονόματα των μεταβλητών δεν είναι υποχρεωτικά, αρκεί ο τύπος τους. Μπορούμε έτσι να γράψουμε ισοδύναμα:

```
void show(char);
```

```
double show(int, float);

int* show(int*, double);
```

Ένα συνηθισμένο λάθος είναι ο τύπος να δηλώνεται μια φορά, όταν οι παράμετροι έχουν τον ίδιο τύπο. Π.χ. το παρακάτω είναι λάθος:

```
void test(int a, b, c);
```

Το σωστό είναι:

`void test(int a, int b, int c);` (ή και χωρίς τα ονόματα: `void test(int, int, int);`

14. Ορισμός συνάρτησης

Η γενική μορφή του ορισμού μιας συνάρτησης είναι:

```
τύπος_επιστροφής όνομα_συνάρτησης(λίστα_παραμέτρων)
{
     /* σώμα συνάρτησης */
}
```

Η πρώτη γραμμή πρέπει να ταιριάζει με τη δήλωση της συνάρτησης με τη διαφορά ότι δεν προστίθεται το ερωτηματικό στο τέλος της. Ο κώδικας (σώμα) της συνάρτησης περιέχει δηλώσεις μεταβλητών και εντολές και βρίσκεται ανάμεσα σε άγκιστρα. Εκτελείται μόνο στην περίπτωση που η συνάρτηση κληθεί από κάποιο σημείο του προγράμματος. Η εκτέλεση τερματίζει αν κληθεί μια εντολή τερματισμού (π.χ. `return`) ή όταν εκτελεστεί η τελευταία εντολή της. Ας δούμε ένα παράδειγμα:

```
#include <stdio.h>
void test(void); // δήλωση συνάρτησης

int main(void)
{
     test(); // κλήση συνάρτησης
     return 0;
}

void test(void) // ορισμός συνάρτησης
{
     // σώμα συνάρτησης
     printf("In\n");
}
```

Το πρόγραμμα θα εμφανίσει `In`. Η C δεν απαιτεί οι ορισμοί των συναρτήσεων να γίνονται μετά τη `main()`, πολλές φορές αυτό προτιμάται, καθώς διευκολύνεται η ανάγνωση του προγράμματος. Το πρωτότυπο μιας συνάρτησης είναι υποχρεωτικό εφόσον ο ορισμός της βρίσκεται μετά την κλήση της. Μια συνάρτηση δεν μπορεί να ορίζεται μέσα σε κάποια άλλη. Επειδή μια συνάρτηση ενδέχεται να καλεί κάποια άλλη, χρειάζεται προσοχή στη σειρά ορισμού τους.

15. Κλήση συνάρτησης

Η κλήση μιας συνάρτησης που δεν δέχεται παραμέτρους γίνεται γράφοντας το όνομά της ακολουθούμενο από κενές παρενθέσεις. Στη συνάρτηση δεν μεταβιβάζεται κάποια πληροφορία. Δείτε ξανά το προηγούμενο παράδειγμα. Στην περίπτωση που η συνάρτηση δέχεται παραμέτρους, αυτό σημαίνει ότι της μεταβιβάζεται πληροφορία μέσω των ορισμάτων της (arguments). Η διαφορά μεταξύ παραμέτρου και ορίσματος είναι ότι ο όρος παράμετρος αναφέρεται στις μεταβλητές που εμφανίζονται στον ορισμό της συνάρτησης, ενώ ο όρος όρισμα αναφέρεται στις εκφράσεις που περιέχονται στην κλήση της συνάρτησης. Για παράδειγμα, δείτε το παρακάτω πρόγραμμα:

```
#include <stdio.h>

int test(int x, int y);

int main(void)
{
     int sum, a = 10, b = 20;

     sum = test(a, b);
     printf("%d\n", sum);
     return 0;
}

int test(int x, int y)
{
     return x+y;
}
```

Αν επιθυμούμε μια συνάρτηση να μπορεί να αλλάξει την τιμή κάποιου ορίσματος, πρέπει να μεταβιβάσουμε στη συνάρτηση τη διεύθυνση μνήμης του ορίσματος και όχι την τιμή του. Επομένως, αφού η συνάρτηση έχει πρόσβαση στη διεύθυνση μνήμης του ορίσματος, μπορεί να μεταβάλλει την τιμή του. Δείτε π.χ. το ακόλουθο πρόγραμμα:

```
#include <stdio.h>

void test(int *p);

int main(void)
{
```

```
     int i = 10;
     test(&i);
     printf("%d\n", i);
     return 0;
}

void test(int *p)
{
     *p = 20;
}
```

Όταν γίνεται η κλήση της `test()` έχουμε `p = &i`. Αφού η τιμή του `p` είναι ίση με τη διεύθυνση μνήμης του `i`, τότε η `test()` μπορεί να αλλάξει την τιμή του. Έτσι η εντολή `*p = 20` αλλάζει το περιεχόμενο της διεύθυνσης, άρα και την τιμή του `i` από `10` σε `20`, το οποίο και εμφανίζεται από το πρόγραμμα.

16. Κλήση συνάρτησης με παράμετρο πίνακα

Όταν η παράμετρος μιας συνάρτησης είναι ένας μονοδιάστατος πίνακας, γράφουμε το όνομα του πίνακα ακολουθούμενο από αγκύλες. Η συνήθης τακτική είναι να παραλείπουμε το μήκος του πίνακα και οι αγκύλες να είναι κενές. Π.χ.:

```
void test(int arr[]);
```

Όταν μεταβιβάζεται ένας πίνακας σε μια συνάρτηση γράφουμε μόνο το όνομα του πίνακα χωρίς τις αγκύλες. Π.χ.:

```
test (arr);
```

Όταν το όνομα ενός πίνακα μεταβιβάζεται σε μια συνάρτηση, ουσιαστικά χρησιμοποιείται σν δείκτης. Ουσιαστικά μεταβιβάζεται ένας δείκτης στο πρώτο του στοιχείο και όχι ένα αντίγραφο του πίνακα. Αυτό γίνεται για καλύτερη απόδοση. Δείτε πώς γίνεται, στο παρακάτω παράδειγμα:

```
#include <stdio.h>

void test(int arr[]);

int main(void)
{
     int i, a[5] = {10, 20, 30, 40, 50};

     test(a);
     for(i = 0; i < 5; i++)
          printf("%d ", a[i]);
     return 0;
```

```
}

void test(int arr[])
{
     arr[0] = arr[1] = 0;
}
```

Όταν καλείται η `test()`, έχουμε `arr = a = &a[0]`, άρα οι `arr[0]` και `arr[1]` αλλάζουν τις τιμές των δύο πρώτων στοιχείων του πίνακα.