Δείκτες (Pointers) στη Γλώσσα C

1. Τι είναι δείκτης;

Ένας **δείκτης (pointer)** είναι μια μεταβλητή που *αποθηκεύει τη διεύθυνση μνήμης* μιας άλλης μεταβλητής.

Παράδειγμα:

```
int x = 10;
int *p = &x;  // ο p κρατάει τη διεύθυνση της μεταβλητής x
```

- x → κανονική μεταβλητή
- &x → η διεύθυνση μνήμης της x
- p → δείκτης σε ακέραιο
- *p → το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης

Παράδειγμα 1 — Βασική πρόσβαση μέσω δείκτη

```
#include <stdio.h>
int main() {
    int x = 42;
    int *p = &x;
    printf("Τιμή x: %d\n", x);
    printf("Τιμή μέσω p: %d\n", *p);
    return 0;
}
```

Παράδειγμα 2 — Αλλαγή τιμής μέσω δείκτη

```
#include <stdio.h>
int main() {
    int x = 5;
    int *p = &x;
    *p = 99;  // αλλάζουμε το x
    printf("Νέα τιμή x: %d\n", x);
    return 0;
}
```

Παράδειγμα 3 — Δείκτης σε διαφορετική μεταβλητή

```
#include <stdio.h>
int main() {
    int a = 10, b = 20;
    int *p = &a;
    printf("p -> %d\n", *p); // 10
    p = &b;                  // τώρα δείχνει στο b
    printf("p -> %d\n", *p); // 20
    return 0;
}
```

2. Δήλωση και αρχικοποίηση δεικτών

Δήλωση

```
int *p;
float *q;
char *c;
```

Αρχικοποίηση με διεύθυνση:

```
int x = 5;
int *p = &x;
```

Σημαντικό:

Ο τύπος του δείκτη πρέπει να ταιριάζει με τον τύπο της μεταβλητής.

Λάθος:

```
float x = 3.14;
   int *p = &x;
```

Παράδειγμα 4 — Δείκτες διαφορετικών τύπων

```
#include <stdio.h>
int main() {
    int x = 1;
    float y = 2.5;
    char c = 'A';
    int *p1 = &x;
    float *p2 = &y;
    char *p3 = &c;
    printf("%d, %.2f, %c\n", *p1, *p2, *p3);
    return 0;
}
```

Παράδειγμα 5 — Δείκτης που αρχικοποιείται αργότερα

```
#include <stdio.h>
int main() {
    int x = 100;
    int *p;
    p = &x;  // αργότερη αρχικοποίηση
    printf("%d\n", *p);
    return 0;
}
```

Παράδειγμα 6 — NULL pointer

```
#include <stdio.h>
int main() {
    int *p = NULL;
    if (p == NULL)
        printf("Ο δείκτης δεν δείχνει πουθενά\n");
    return 0;
}
```

3. Τελεστής * (dereference)

Ο τελεστής * διαβάζει ή γράφει την τιμή της μεταβλητής στην οποία δείχνει ο δείκτης.

```
int x = 10;
int *p = &x;

printf("%d", *p); // εκτυπώνει 10
```

Αλλαγή τιμής μέσω δείκτη:

```
*p = 20;
printf("%d", x); // εκτυπώνει 20
```

Παράδειγμα 7 — Ανάγνωση και εγγραφή μέσω pointer

```
#include <stdio.h>
int main() {
    int x = 10;
    int *p = &x;
    printf("Τιμή μέσω *p: %d\n", *p);
    *p = *p + 5;   // αυξάνει το x κατά 5
    printf("Νέα τιμή x: %d\n", x);
    return 0;
}
```

Παράδειγμα 8 — Δύο μεταβλητές στο ίδιο pointer

```
#include <stdio.h>
int main() {
    int a = 3, b = 9;
    int *p = &a;
    printf("p-> %d\n", *p);
    p = &b;  // ο p δείχνει στη b
    printf("p-> %d\n", *p);
    return 0;
}
```

4. Δείκτες και μνήμη

Κάθε μεταβλητή αποθηκεύεται σε κάποια θέση μνήμης. Οι δείκτες χειρίζονται αυτές τις διευθύνσεις.

```
int x = 7;
int *p = &x;
printf("%p", p); // εκτυπώνει π.χ. 0x7ffeefbff45c
```

Παράδειγμα 9 — Εκτύπωση διευθύνσεων

```
#include <stdio.h>
int main() {
    int x = 10;
    int *p = &x;
    printf("Διεύθυνση x: %p\n", &x);
    printf("Τι κρατάει ο p (διεύθυνση): %p\n", p);
    return 0;
}
```

Παράδειγμα 10 — Δύο μεταβλητές, δύο δείκτες

```
#include <stdio.h>
int main() {
    int a = 100, b = 200;
    int *p1 = &a;
    int *p2 = &b;
    printf("a: %p, b: %p\n", p1, p2);
    return 0;
}
```

Παράδειγμα 11 — Δείκτης αλλάζει διεύθυνση

```
#include <stdio.h>
int main() {
    int x = 5, y = 7;
    int *p = &x;
    printf("p -> %p\n", p);
    p = &y;
    printf("p -> %p\n", p);
    return 0;
}
```

5. Δείκτες και πίνακες

Οι πίνακες «γλιστρούν» σε δείκτες.

```
int arr[3] = {10, 20, 30};
int *p = arr; // ισοδύναμο με int *p = &arr[0];
```

Πρόσβαση:

```
printf("%d", *(p + 1)); // 20
```

Ισοδύναμο με:

```
printf("%d", arr[1]); // 20
```

Παράδειγμα 12 — Array → pointer

```
#include <stdio.h>
int main() {
    int arr[3] = {10, 20, 30};
    int *p = arr;
    printf("%d\n", *p);         // 10
    printf("%d\n", *(p + 1));   // 20
    printf("%d\n", *(p + 2));   // 30
    return 0;
}
```

Παράδειγμα 13 — Αλλαγή στοιχείων πίνακα μέσω δείκτη

```
#include <stdio.h>
int main() {
    int arr[3] = {1, 2, 3};
    int *p = arr;
    *(p + 1) = 100;
    printf("%d %d %d\n", arr[0], arr[1], arr[2]);
    return 0;
}
```

Παράδειγμα 14 — Ο δείκτης κινείται μέσα στον πίνακα

```
#include <stdio.h>
int main() {
    int arr[4] = {5, 10, 15, 20};
    int *p = arr;
    for (int i = 0; i < 4; i++) {
        printf("arr[%d] = %d\n", i, *p);
        p++;  // προχώρα στο επόμενο στοιχείο
    }
    return 0;
}
```

6. Αριθμητική δεικτών (Pointer arithmetic)

Οι δείκτες μπορούν να αυξάνονται/μειώνονται σύμφωνα με το μέγεθος του τύπου τους.

```
int *p = arr;
p++;          // τώρα δείχνει στο arr[1]
```

Αν int = 4 bytes → το p++ μετακινείται κατά 4 bytes.

Παράδειγμα 15 — p++, p--

```
#include <stdio.h>

int main() {
    int arr[3] = {10, 20, 30};
    int *p = arr;
    printf("%d\n", *p);   // 10
    p++;   // δείχνει στο arr[1]
    printf("%d\n", *p);   // 20
    p++;   // δείχνει στο arr[2]
    printf("%d\n", *p);   // 30
    p--;   // πίσω στο arr[1]
    printf("%d\n", *p);   // 20
    return 0;
}
```

Παράδειγμα 16 — Διαφορά δύο pointer

```
#include <stdio.h>
int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int *p1 = arr;
    int *p2 = &arr[4];
    int distance = p2 - p1; // σε στοιχεία, όχι σε bytes
    printf("Απόσταση: %d στοιχεία\n", distance); // 4
    return 0;
}
```

Παράδειγμα 17 — Άλμα δύο θέσεων

```
#include <stdio.h>
int main() {
    int arr[] = {5, 10, 15, 20};
    int *p = arr;
    p += 2; // πήγαινε 2 στοιχεία μπροστά
    printf("%d\n", *p);  // 15
    return 0;
}
```

7. Δείκτες και συναρτήσεις

Οι δείκτες επιτρέπουν την **πέραση μεταβλητών με αναφορά (by reference)**.

Παράδειγμα αλλαγής τιμής μέσα από συνάρτηση:

```
void change(int *p) {*p = 100;}
int main() {
    int x = 5;
    change(&x);
    printf("%d", x);   // εκτυπώνει 100
}
```

Παράδειγμα 18 — Αλλαγή μεταβλητής από συνάρτηση

```
#include <stdio.h>
void setZero(int *p) {
    *p = 0;
}
int main() {
    int x = 10;
    setZero(&x);
    printf("%d\n", x);  // 0
    return 0;
}
```

Παράδειγμα 19 — Swap δύο τιμών

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 5, y = 10;
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

Παράδειγμα 20 — Συνάρτηση που αυξάνει valeur κατά n

```
#include <stdio.h>
void add(int *p, int n) {
    *p += n;
}
int main() {
    int x = 7;
    add(&x, 5); // x = 12
    printf("%d\n", x);
    return 0;
}
```

8. Δείκτες σε δείκτες (double pointers)

Χρησιμεύουν σε δυναμικούς πίνακες / συναρτήσεις που πρέπει να τροποποιήσουν δείκτες

```
int x = 10;
int *p = &x;
int **pp = &p;
printf("%d", **pp); // 10
```

Παράδειγμα 21 — Βασικό double pointer

```
#include <stdio.h>
int main() {
    int x = 10;
    int *p = &x;
    int **pp = &p;
    printf("%d\n", **pp);
    return 0;
}
```

Παράδειγμα 22 — Αλλαγή τιμής μέσω double pointer

```
#include <stdio.h>
int main() {
    int x = 5;
    int *p = &x;
    int **pp = &p;
    **pp = 50;
    printf("%d\n", x);  // 50
    return 0;
}
```

Παράδειγμα 23 — Αλλαγή του ίδιου του pointer

```
#include <stdio.h>
int main() {
    int a = 100, b = 200;
    int *p = &a;
    int **pp = &p;
    *pp = &b; // αλλάζει ο p ώστε να δείχνει στο b
    printf("%d\n", *p); // 200
    return 0;
}
```

Παράδειγμα 24 — Συνάρτηση που αλλάζει pointer

```
#include <stdio.h>
void changePointer(int **pp, int *newAddress) {
    *pp = newAddress; // η pp αλλάζει τον p
}

int main() {
    int a = 10, b = 20;
    int *p = &a;
    changePointer(&p, &b);
    printf("%d\n", *p);  // 20
    return 0;
}
```

9. Δυναμική μνήμη (malloc, free)

Η malloc() δεσμεύει μνήμη *κατά τον χρόνο εκτέλεσης*.

```
int *p = malloc(sizeof(int));
*p = 50;

printf("%d", *p);

free(p);  // αποδέσμευση μνήμης
```

10. Συνήθη λάθη με δείκτες

Uninitialized pointer

```
int *p; // περιέχει τυχαία διεύθυνση!!!
*p = 10; // ΚΡΙΤΙΚΟ ΛΑΘΟΣ
```

Dangling pointer

```
int *p = malloc(sizeof(int));
free(p);
*p = 10; // λάθος, η μνήμη έχει αποδεσμευτεί
```

Memory leak

```
int *p = malloc(100 * sizeof(int));
// ξεχάσαμε free(p);
```

Συνοπτικό παράδειγμα – Όλα μαζί

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

void multiplyByTwo(int *n) {
    *n = *n * 2;
}

int main() {
    int x = 10;
    int *p = &x;

    printf("x = %d\n", x);
    printf("address of x = %p\n", p);
    printf("value via pointer = %d\n", *p);

    multiplyByTwo(&x);
    printf("after function: x = %d\n", x);

    // Dynamically allocate memory for 5 integers
    int *arr = malloc(5 * sizeof(int));
    for (int i = 0; i < 5; i++)
        arr[i] = i + 1;

    printf("arr[2] = %d\n", *(arr + 2));

    free(arr);

    return 0;
}
```