CURS 02 – PP
ALOCAREA DINAMICĂ A MEMORIEI

Întrebare (din interviuri):
În limbajul C, un tablou unidimensional este **echivalent** cu un pointer?

Răspuns:
Depinde de context!!!

1) Dacă tabloul este parametrul unei funcții, atunci răspunsul este DA!!!

```
void afisare(int v[100], int n)
void afisare(int v[], int n)
void afisare(int *v, int n)
```

Dacă se cunoaște adresa primului element din tablou, echivalentă/egală cu numele tabloului, atunci se poate accesa orice element al tabloului!
Dimensiunea efectivă a tabloului (parametrul n) este necesară pentru a ști câte elemente sunt efectiv utilizate și/sau pentru a evita ieșirea din tablou.

2) (Generalizare) Dacă dorim doar să accesăm elementele unui tabloul deja alocat (static sau dinamic), atunci putem să realizăm acest lucru printr-un pointer, deci răspunsul este DA!!!

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

int main()
{
    int n = 10, i;
    int v[100], *p;
    //Tabloul v poate fi alocat și dinamic!

    p = v;
    for(i = 0; i < n; i++)
    {
        *p = i+1;
        p++;
    }
```

```
    printf("Tabloul:\n");
    p = v;
    for(i = 0; i < n; i++)
    {
        printf("%d ", *p);
        p++;
    }

    printf("\n");

    return 0;
}
```

3) Dacă tabloul NU este deja alocat (static sau dinamic), atunci răspunsul este NU!!!

<table>
<tr><td colspan="2"><code>int v[100];</code></td><td><code>int *v;</code></td><td rowspan="3">NU
(din punct de vedere funcțional)</td></tr>
<tr><td colspan="2">sizeof(v) = 100*sizeof(int) = 400 octeți</td><td>sizeof(v) = 4 octeți</td></tr>
<tr><td colspan="2">În v se pot memora 100 de numere întregi.</td><td>În v se poate memora o adresă.</td></tr>
<tr><td><code>int v[100];</code></td><td colspan="2"><code>int *v;
v=(int*)malloc(100*sizeof(int))
...
free(v);</code></td><td rowspan="2">DA
(din punct de vedere funcțional)</td></tr>
<tr><td colspan="3">În v se pot memora 100 de numere întregi.</td></tr>
</table>

<table>
<tr><td><code>int v[100];</code></td><td><code>int *v;
v=(int*)malloc(100*sizeof(int))
...
free(v);</code></td><td rowspan="2">NU
(din punct de vedere al memoriei)</td></tr>
<tr><td>v este memorat în zona STACK (stivă) și are management automat al memoriei</td><td>v este memorat în STACK, dar accesează o zonă de memorie din HEAP și NU are management automat al memoriei</td></tr>
</table>

ERORI SPECIFICE OPERAȚIILOR CU POINTERI

a) **Wild pointer** = accesare unei zone de memorie a cărei adresă NU a fost obținută corect (i.e., fie ca adresă a unei variabile existente cu operatorul &, fie folosind alocarea dinamică)!

#include <stdio.h>

```
int main()
{
    int *t;

    //in variabila t este memorata o valoare reziduala,
    //dar care este considerata ca fiind
    //o adresa de memorie
    *t = 1000;

    printf(" t = %p\n", t);
    printf("*t = %d\n", *t);

    return 0;
}
```

Programul de mai sus poate funcționa uneori, dar poate altera o zonă de memorie a altui program!!!

Rezolvare:

```
int *t = NULL;
```

În acest caz, programul va furniza întotdeauna o eroare la rulare!!!

b) **Dangling pointer** = accesare unei zone de memorie a cărei adresă a fost corectă la un moment dat, dar NU mai este corectă acum (i.e., zona de memorie respectivă a fost eliberată între timp)!

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

int main()
{
    int *t;
```

```
    t = (int*)malloc(sizeof(int));
    *t = 1000;

    printf(" t = %p\n", t);
    printf("*t = %d\n", *t);

    free(t);

    *t = -17;
    printf("\n t = %p\n", t);
    printf("*t = %d\n", *t);

    //......

    //nu stiu sigur daca *t mai contine valoarea -17!!!
    int x = *t + 1;

    return 0;
}
```

Programul de mai sus funcționează aproape întotdeauna, dar zona de memorie în care se păstrează valoarea -17 poate fi alterată de alte programe!!!

Rezolvare:

```
free(t);
t = NULL;
```

În acest caz, programul va furniza întotdeauna o eroare la rulare!!!

Exemplu:

Scrieți un program care citește de la tastatură un tablou unidimensional format din *n* numere întregi, iar apoi construiește un tablou format din numerele strict negative din tabloul inițial și un alt tablou format din numerele strict pozitive. Toate cele 3 tablouri vor fi alocate dinamic!

Varianta 1 (spațiul de memorie utilizat este aproximativ egal cu dublul spațiului de memorie necesar tabloului inițial):

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

```
void afisareVector(int *vect, int n) {
    int index;
    printf("\nElementele tabloului sunt: ");
    for (index = 0; index < n; index++)
        printf("%d ", *(vect + index));
    printf("\n");
}

int main(){
    int n, index, size_poz = 0, size_neg = 0, *v = NULL, *poz = NULL,
     *neg = NULL, *aux = NULL;
    printf("Introduceti n: ");
    scanf("%d", &n);
    v = (int*)malloc(n * sizeof(int));

    printf("Introduceti elementele tabloului: ");
    for (index = 0; index < n; index++)
        scanf("%d", v + index);

    for (index = 0; index < n; index++) {
        if (*(v + index) > 0) {
            aux = (int*)realloc(poz, (size_poz + 1) * sizeof(int));
            if (aux != NULL) {
                poz = aux;
                *(poz + size_poz) = *(v + index);
                size_poz++;
            }
            else {
                printf("\nMemorie insuficienta!\n");
                free(v);
                free(poz);
                free(neg);
                exit(0);
            }
        }
        else if (*(v + index) < 0){
            aux = (int*)realloc(neg, (size_neg + 1) * sizeof(int));
            if (aux != NULL) {
                neg = aux;
                *(neg + size_neg) = *(v + index);
                size_neg++;
            }
            else {
                printf("\nMemorie insuficienta!\n");
                free(v);
                free(poz);
                free(neg);
                exit(0);
            }
```

```
        }
    }

    free(v);

    if (size_poz) {
        printf("\n----- Vectorul cu numere strict pozitive -----");
        afisareVector(poz, size_poz);
    }
    else
        printf("\nNu exista numere strict pozitive!\n");
    free(poz);

    if (size_neg) {
        printf("\n----- Vectorul cu numere strict negative -----");
        afisareVector(neg, size_neg);
    }
    else
        printf("\nNu exista numere strict negative!\n");
    free(neg);

    return 0;
}
```

Varianta 2 (spațiul de memorie utilizat este aproximativ egal cu spațiului de memorie necesar tabloului inițial <=> memorie constantă):

Vom parcurge tabloul inițial de la sfârșit spre început și, după ce adăugăm elementul curent într-unul dintre cele două tablouri cu elemente strict pozitive sau strict negative, îl ștergem folosind realloc.

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

void afisareVector(int *vect, int n) {
    int index;
    printf("\nElementele tabloului sunt: ");
    for (index = 0; index < n; index++)
        printf("%d ", *(vect + index));
    printf("\n");
}

int main(){
    int n, index, size_poz = 0, size_neg = 0, *v = NULL, *poz =
NULL, *neg = NULL, *aux = NULL;
    printf("Introduceti n: ");
    scanf("%d", &n);
    v = (int*)malloc(n * sizeof(int));
```

```
printf("Introduceti elementele tabloului: ");
for (index = 0; index < n; index++)
    scanf("%d", v + index);

for (index = n - 1; index >= 0; index--) {
    if (*(v + index) > 0) {
        aux = (int*)realloc(poz, (size_poz + 1) * sizeof(int));
        if (aux != NULL) {
            poz = aux;
            *(poz + size_poz) = *(v + index);
            size_poz++;
        }
        else {
            printf("\nMemorie insuficienta!\n");
            free(v);
            free(poz);
            free(neg);
            exit(0);
        }
    }
    else if (*(v + index) < 0){
        aux = (int*)realloc(neg, (size_neg + 1) * sizeof(int));
        if (aux != NULL) {
            neg = aux;
            *(neg + size_neg) = *(v + index);
            size_neg++;
        }
        else {
            printf("\nMemorie insuficienta!\n");
            free(v);
            free(poz);
            free(neg);
            exit(0);
        }
    }
    v = (int *)realloc(v, (n - 1) * sizeof(int));
    n--;
}

if (size_poz) {
    printf("\n----- Vectorul cu numere strict pozitive -----");
    afisareVector(poz, size_poz);
}
else
    printf("\nNu exista numere strict pozitive!\n");
free(poz);

if (size_neg) {
```

```
        printf("\n----- Vectorul cu numere strict negative -----");
        afisareVector(neg, size_neg);
    }
    else
        printf("\nNu exista numere strict negative!\n");
    free(neg);

    return 0;
}
```