# Απαλλακτική Εργασία στις Δομές Δεδομένων

Version: 2023-03-07

# Table of Contents

Θέμα 1 . . . . . 1
Θέμα 2 . . . . . 3

# Θέμα 1

**Υποθέτουμε ότι A, B, C είναι πίνακες με δείκτη 1..10 και τύπο στοιχείων «πραγματικούς αριθμούς». Να γραφεί μια διαδικασία C η οποία χρησιμοποιεί λειτουργίες retrieve και update για να υλοποιήσει την πρόσθεση πινάκων A := B + C.**

Αρχικά θα υλοποιούμε την `update(S,c,i)` και `retrieve(S,c,i)` όπως παρουσιάζονται στις σημειώσεις, δηλαδή όπου `S` ο πίνακας, `c` το προς εισαγωγή ή προς επιστροφή στοιχείο και `i` ο δείκτης στον πίνακα.

```
void update(float S[], float c, int i) {
    *(S+i) = c;
}

void retrieve(float S[], float *c, int i) {
    *c = S[i];
}
```

Στην `update` εκμεταλλευόμαστε ότι το όνομα του πίνακα είναι αναφορά στη θέση μνήμης του για να αλλάξουμε απευθείας τη τιμή του ζητούμενου στοιχείου. Στην `retrieve`, δεδομένου ότι θέλουμε να έχουμε πρόσβαση από τη `main` στο επιστρεφόμενο στοιχείο αλλά και να κρατήσουμε την υπογραφή της συνάρτησης κοντά στης δοθείσης, αντί απλά ενός `float c` στέλνουμε τη θέση μνήμης μιας `float` τιμής.

```
#include <stdio.h>

int main() {
    float A[10];
    float B[10];
    float C[10];

    float entry;
    float retrievedB,retrievedC;

    for (int i = 0; i<10; i++) {
        printf("Give element %d of B: ",i+1);
        scanf("%f",&entry);
        update(B,entry,i);
        printf("Give element %d of C: ",i+1);
        scanf("%f",&entry);
        update(C,entry,i);

        retrieve(B,&retrievedB,i);
        retrieve(C,&retrievedC,i);
        update(A,retrievedB+retrievedC,i);
    }

    float retrievedA;
```

```
    for (int i = 0; i<10; i++) {
        retrieve(A, &retrievedA, i);
        printf("Element %d of A:=B+C is %f\n",i+1,retrievedA);
    }

    return 0;
}
```

Στη `main` ορίζουμε τους τρεις πίνακες 10 θέσεων, κάνουμε `update` τις τιμές των δύο με εισόδους από τον χρήστη και έπειτα κάνουμε `update` τις τιμές του τελικού πίνακα, μέσω αλλεπάλληλων `retrieve` των αντιστοίχων στοιχείων των πρώτων πινάκων.

# Θέμα 2

**Να υπολογιστεί η διεύθυνση κάθε στοιχείου ενός πίνακα A(1:2,1:3,3:3,1:2). Θεωρείστε ότι ο πίνακας έχει βασική διεύθυνση b=100 και μήκος συνιστώσας L=6, ενώ τα άνω και κάτω όρια των δεικτών του είναι όπως παραπάνω.**

Θα υλοποιήσουμε τη συνάρτηση απεικόνισης πίνακα για διάσταση πίνακα και όρια δοσμένα από τον χρήστη.

```
#include <stdbool.h>

/*global variable that holds the dimension of the user's array*/
int dim;

/* a pointer to a vector of pointers each one of which point to the
 lower and upper bound indexes of the user's array */
int** boundsPtr;

bool getUserInput(){
    int i,j;

    printf("Type the array's dimension: ");
    scanf("%d", &dim);

    /* user input validation */
    if (dim <= 0){
        printf("Invalid input. \n");
        return false;
    }
    else{
        /* dynamic allocation of the pointer (the rows of the 2D array
representation)*/
        boundsPtr = (int**)malloc(sizeof(int)*dim);
        for(i=0;i<dim;i++){
            /* dynamic allocation of a pointer (the columns of the 2D array
representation)*/
            boundsPtr[i] = (int*)malloc(sizeof(int)*2);
            for (j=0;j<2;j++){
                printf("Type index %d of dimension %d\n",j+1,i+1);
                scanf("%d",&boundsPtr[i][j]);
                if (j%2 == 1){
                    /* user input validation */
                    if (boundsPtr[i][j-1]>boundsPtr[i][j]){
                        printf("invalid input.\n");
                        return false;
                    }
                }
            }
```

```
        }
        return true;
    }
}
```

Στην `getUserInput()` ζητάμε από τον χρήστη τη διάσταση του πίνακα και, αν είναι θετική, ορίζουμε τον διδιάστατο πίνακα (`dim` γραμμές, 2 στήλες) που θα κρατήσει τα όρια που θα δώσει ο χρήστης, κάνοντας έναν υποτυπώδη έλεγχο εγκυρότητάς τους.

Αντιμετωπίζουμε τον διδιάστατο πίνακα σαν πίνακα με δείκτες για άλλους μονοδιάστατους πίνακες (*δείκτες δεικτών*), για να είναι εύκολη η δήλωσή του σαν `global` μεταβλητή ενώ δεν ξέρουμε εκ των προτέρων το μέγεθός του. Αντίστοιχη τακτική χρησιμοποιούμε αργότερα και για τον `indexes`, που θα έχει όλους τους διαφορετικούς συνδυασμούς συντεταγμένων βάσει των ορίων που έχουν δοθεί.

```
/* 2D array that will store all valid coordinates*/
int** indexes;

/* find total number of elements of multidimensional array */
int findTotal() {
    int total = 1;
    for (int i = 0; i < dim; i++) {
        total *= (boundsPtr[i][1] - boundsPtr[i][0] + 1);
    }
    return total;
}

/* find recursively the valid indexes of all the elements of the user's array and
calculate their address*/
int count = 0; // global counter for entries in indexes, to avoid unexpected behaviour
during recursion
void findIndexes(int guide, int temp[]) {
    for (int i = boundsPtr[guide][0]; i <= boundsPtr[guide][1]; i++) {
        temp[guide] = i;
        if (guide == dim - 1) {
            for (int j = 0; j < dim; j++) {
                indexes[count][j] = temp[j];
            }
            count++;
        } else {
            findIndexes(guide+1, temp);
        }
    }
}
```

Με την `findTotal` υπολογίζουμε τον αθροιστικό αριθμό στοιχείων που θα έχει ο πολυδιάστατος πίνακας, έτσι ώστε να μπορέσουμε έπειτα με την `findIndexes` να βρούμε τι συντεταγμένες μπορεί να έχει ένα στοιχείο του πίνακα. Στην περίπτωση της εκφώνησης, όλες τις πιθανές τετράδες

συντεταγμένων $(i_1, i_2, i_3, i_4)$ με

- $1 <= i_1 <= 2$
- $1 <= i_2 <= 3$
- $i_3 = 3$
- $1 <= i_4 <= 2$

```
/* calculate all the (dimension + 1) coefficients of the given element of the array */
void findCoefficients(int coefficients[]) {
    coefficients[dim] = length;

    for (int i = dim - 1; i > 0; i--) {
        coefficients[i] = (boundsPtr[i][1] - boundsPtr[i][0] + 1) * coefficients[i+1];
    }

    coefficients[0] = base;
    for (int i = 0; i < dim; i++) {
        coefficients[0] -= coefficients[i+1] * boundsPtr[i][0];
    }
}

/* calculate and print the exact memory address of the given element of the array */
void findAddress(int total, int coefficients[]) {
    int addr;
    for (int i = 0; i < total; i++) {
        printf("Address of element ( ");
        addr = coefficients[0];
        for (int j = 0; j < dim; j++) {
            printf("%d ", indexes[i][j]);
            addr += coefficients[j + 1] * indexes[i][j];
        }
        printf(") is %d\n", addr);
    }
}
```

Στην `findCoefficients` υπολογίζουμε όλους τους συντελεστές $c_i$, `0 <= i <= 4` βάσει του τύπου και στην `findAddress`, έχοντας ήδη γεμίσει των πίνακα συντεταγμένων `indexes`, βρίσκουμε τις διευθύνσεις κάθε στοιχείου.

```
int main() {
    if (getUserInput()){
        for (int i=0; i<dim; i++) {
            printf("===Row %d===\n",i+1);
            printf("Lower bound: %d, Upper bound: %d\n",boundsPtr[i][0],boundsPtr[i][
1]);
            printf("\n");
        }
```

```
        int total = findTotal();

        /* initialize recursion */
        int coefficients[dim+1];
        findCoefficients(coefficients);

        indexes = (int**) malloc(sizeof(int) * 100);
        for (int i = 0; i < total; i++) {
            indexes[i] = (int*) malloc(sizeof(int) * dim);
        };
        int temp[dim];

        findIndexes(0, temp);
        findAddress(total, coefficients);
    }
    return 0;
}
```

```
Address of element ( 1 1 3 1 ) is 100
Address of element ( 1 1 3 2 ) is 106
Address of element ( 1 2 3 1 ) is 112
Address of element ( 1 2 3 2 ) is 118
Address of element ( 1 3 3 1 ) is 124
Address of element ( 1 3 3 2 ) is 130
Address of element ( 2 1 3 1 ) is 136
Address of element ( 2 1 3 2 ) is 142
Address of element ( 2 2 3 1 ) is 148
Address of element ( 2 2 3 2 ) is 154
Address of element ( 2 3 3 1 ) is 160
```