

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Кафедра теоретичних основ радіотехніки

## **ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №4**

**« Робота з двовимірними динамічними масивами »**

з дисципліни: «Інформатика 1»

	Виконав: Луцкевич Володимир Андрійович  Група: РЕ-12  Викладачі: доцент Катін П.  Оцінка: _____  Підпис: _____
--	---

Київ – 2021

**Мета:** скласти програму для роботи з двовимірними масивами.

Код:

```
#include <stdio.h>
#include <conio.h>
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include < malloc.h > // для використання функцій динамічного розподілу пам'яті
#include <conio.h> // для створення текстовоо інтерфесу
#include <time.h>

#define size 8 //розмір масиву

void printArray( int[], int); // за допомогою цієї функції виводимо значення заданого масиву
int search_Max(int[], int); // функція в якій вказано масив і розмір масиву, повертає максимально знайдене значення
цього масиву
int search_Min(int[], int);

void search_Max2(int[], int, int * pMax, int * pIndex); // pMax- зберігає адресу змінної Max2, pIndex- зберігає адресу
змінної index

void sort_1D(int *z, int S); //сортування одномірного масиву z, S - розмір масиву

int ArrayBFunc(int[size], int); // масив складається із 4 елементів

//void printArray2(int*, int);

//void printArray3(int*, int);
```

```

int main()

{

    int num1;
    //_____
    int* a; // вказівник на масив
    int i, j, n, m; // індекси: a - вказівник на масив, m - кількість стовпців i - індекс рядка j індекс стовпця
    system("chcp 1251");
    system("cls");
    printf("Number of lines: ");
    scanf("%d", &n);
    printf("Number of columns: ");
    scanf("%d", &m);

    //malloc - функція для визначення розміру масиву в байтах
    // int sizeof() - для точного визначення розміру елементу
    // виділення пам'яті
    a = (int*)malloc(n * m * sizeof(int)); // n*m-(розмір елементу)- об'єм пам'ті необхідний для розміщення
двовимірного масиву

    //printf("%d Розмір масиву\n\t = ", a);

    // ввести кожен елемент масиву
    for (i = 0; i < n; i++) // цикл по рядкам
    {
        for (j = 0; j < m; j++) // цикл по стовпцям
        {
            printf("a[%d][%d] = ", i, j); //index = i*m+j;
            scanf("%d", (a + i * m + j)); // a - вказівник на масив, m - кількість стовпців i - індекс рядка j
індекс стовпця
        }
    }
    // вивести кожен елемент масиву
    for (i = 0; i < n; i++) // цикл по рядкам
    {
        for (j = 0; j < m; j++) // цикл по стовпцям
        {
            /*(a + i * m + j) звернення до елементу index = i*m+j; // кожен елмен
            printf("%5d ", *(a + i * m + j)); // поле шириною 5 символів під елмент масиву
        }
        printf("\n");
    }

    int ArrayA = *(a + i * m + j);

    num1 = *(a + 0 * m + 1); // змінюючи i та j можна викликати будь який елемент масиву

    printf("\nTransposed matrix A\n");
    for (i = 0; i < n; i++) // ідентична частина, але результат виводиться в консоль
    {
        for (j = 0; j < m; j++)

        {
            int ArrayAT = *(a + j * m + i);
            printf("\t %d ", ArrayAT);

        }
    }
}

```

```

        printf("\n");
    }

    printf("\t\n");

    //printf("ЧИСЛО = \n%5d  ", num1);

//    free(a);
//    getchar(); getchar();
//    -----

    int* b; // вказівник на масив

    // виділення пам'яті
    // malloc - функція для визначення розміру масиву в байтах
    // int sizeof() - для точного визначення розміру елементу
    // виділення пам'яті
    b = (int*)malloc(n * m * sizeof(int)); // n·m·(розмір елементу) - об'єм пам'ті необхідний для розміщення
    двовимірного масиву

    //printf("%d Розмір масиву\n\t = ", a);
    // ввести кожен елемент масиву

    for (i = 0; i < n; i++) // цикл по рядкам
    {
        for (j = 0; j < m; j++) // цикл по стовпцям
        {
            printf("b[%d][%d] = ", i, j); // index = i*m+j;
            scanf_s("%d", (b + i * m + j)); // a - вказівник на масив, m - кількість стовпців i - індекс рядка j
            індекс стовпця
        }
    }
    printf("\nmatrix B\t\n ");
    // ввести кожен елемент масиву
    for (i = 0; i < n; i++) // по рядкам
    {
        for (j = 0; j < m; j++) // цикл по стовпцям
        {
            /*(a + i * m + j) звернення до елементу index = i*m+j; // кожен елмен
            printf("%5d ", *(b + i * m + j)); // поле шириною 5 символів під елмент масиву
        }
        printf("\n ");
    }
    int ArrayB = *(b + i * m + j);

```

```

//Множення матриць
//(ArrayA) (ArrayB);
int* c;
c = (int*)malloc(n * m * sizeof(int));
int k;
for (i = 0; i < n; i++)// ідентична частина, але результат виводиться в консоль
{
    for (j = 0; j < m; j++)

    {

        for (k = 0; k < m; k++) // кількість стовпців і рядків однакова
        {
            *(c + i * m + j) = 0;
            /*int tmp2 = *(b + k * m + j);
            int tmp3 = *(a + i * m + k);*/

            *(c + i * m + j) += *(a + k * m + i) * *(b + k * m + j);
            /**(c + i * m + j) = *(a + i * m + k) * *(b + k * m + j);*/
            //int nk = *(a + k * m + i) * *(b + k * m + j);
            //printf("\n%d - [%d][%d]", &nk, i, j);
            // printf("\n%d - [%d][%d]", *(c + i * m + j), i, j);

        }

    }

}

printf("\nmultiplication Matrix Result A * B( A transpoed )\n");
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
        printf("%5dc ", *(c + i * m + j));
    printf("\n");
}

/*int ArrayAxB=*(c + 1 * m + 1) = *(a + 1 * m + 1) * *(b + 1 * m + 1);
printf("%5dC ", ArrayAxB);*/
//-----
// Додавання матриць

int* d;
d = (int*)malloc(n * m * sizeof(int));

printf("\n sum of matrices A + B = \n");
for (i = 0; i < n; i++)// ідентична частина, але результат виводиться в консоль
{
    for (j = 0; j < m; j++)

    {
        *(d + i * m + j) = *(a + i * m + j) + *(b + i * m + j);
        printf("\t %5d ", *(d + i * m + j));
    }
    printf("\n");
}

```

```

printf("\t\n");
// _____

// блок пошуку макимального та мінімального елемента матриці A

int MinNum, MaxNum;
MinNum = MaxNum = *(a + 0 * m + 0);
for (i = 0; i < n; i++)
{
    // проходимо кожний стовпчик строки i
    for (j = 0; j < m; j++)
    {
        //перевіряємо кожен елемент масива з максимумом
        if (*(a + j * m + i) > MaxNum)
        {
            MaxNum = *(a + j * m + i);
        }

        if (*(a + j * m + i) < MinNum)
        {
            MinNum = *(a + j * m + i);
        }
    }
}

printf("\t\nThe maximum element of the matrix A - %d\n", MaxNum); //вивести максимальний елемент

printf("\t\n smallest element of the matrix A - %d\n", MinNum); //вивести найменший елемент

// _____

//блок сортування матриць за зростанням
int sort;

printf("\n\tMatrix sorting A za zrostaniam ");
sort = *(a + j * m + i);
for (int k = 0; k < n * m; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (j != n - 1) {
                if (*(a + (j + 1) * m + i) < *(a + j * m + i)) {
                    int tmp = *(a + (j + 1) * m + i);
                    *(a + (j + 1) * m + i) = *(a + j * m + i);
                    *(a + j * m + i) = tmp;
                }
            }
        }
    }
}

```

```

        else {
            if ((*a + 0 * m + (i + 1)) < *(a + j * m + i)) && (i != n - 1)) {
                int tmp = *(a + 0 * m + (i + 1));
                *(a + 0 * m + (i + 1)) = *(a + j * m + i);
                *(a + j * m + i) = tmp;
            }
        }
    }
}

for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j)
        printf("\t%d", *(a + j * m + i));

}

printf("\t\n\n");

// _____

```

int z[size] = {24, 12, 4, 1, 30, 12, 23, 4}; // всі значення які задаються для кожного елементу масиву

int max, min, Max2, index; // Max2 зберігає максимальне значення елементу масива і index індекс цього елементу

printArray(z, size); // за допомогою функцій printArray, виконується виклик функції z

max = search\_Max(z, size);

printf("\n\nmax=%d", max); //вивід змінної max

min = search\_Min(z, size);

printf("\n\nmin=%d", min); //вивід змінної min

search\_Max2(z, size, &Max2, &index);

printf("\n\nMax2=%d index=%d \n\n", Max2, index);

sort\_1D(z, size); // для сортування масиву

printf("\n\nSorted array:\n"); //виведення відсортованого масиву

```

    printArray(z, size);
    printf("\n");

    //printArray2( z, size); // z адреса першого елемента масиву, зберігається в імені масиву, size розмір масиву
    //printf("\n");

    //printArray3(z, size); // z адреса першого елемента масиву, зберігається в імені масиву, size розмір масиву
    //printf("\n");

    //int y[size] = { 1, 8, 12, 15 }; // всі елементи після 4 масиву будуть автоматично обнулятися
    //int r[size] = { 0 }; //обнулення всіх елементів масиву
    //int t[size] = { 4 };// лише перший елемент масиву отримає значення 4, всі інші будуть = 0
        return 0;
    }

//_____

void printArray(int *Z, int S ) // S локальна змінна яка отримує значення size
{
    int j;
    for (j = 0; j <= S - 1; j++) { // контроль щоб не виходити за межі масиву
        //for (j = S-1; j >= 0; j--)

            printf("%4d", Z[j] );

        }
    }

//_____

int search_Min( int *z, int S)
{
    int temp_min = z[0]; //змінна temp_max отримує значення першого елемету масиву

    int j;

    for (j = 1; j <= S - 1; j++) { // контроль щоб j не виходив за межі масиву, максимальне значення для j це S-1

        if (z[j] < temp_min) { //перевірка якщо на черговий елемент масиву z через індекс j
            temp_min = z[j];
            printf("\ntemp_max=%d", temp_min); // вивід значення змінної temp_max для кожного
значення
            //getch();

        }
    }
    return temp_min; //повернутися в точку виклику search_Max(z, size);
}

//_____

int search_Max(int *z, int S)
{
    int temp_max = z[0]; //змінна temp_max отримує значення першого елемету масиву

    int j;

```



```

        for (j = 1; j <= S - 1; j++) { // контроль щоб j не виходив за межі масиву, максимальне значення для j це S-1

            if (z[j] > temp_max) { //перевірка якщо на черговий елемент масиву z через індекс j
                temp_max = z[j];
                printf("\ntemp_max=%d", temp_max); // вивід значення змінної temp_max для кожного
значення
                //getch();
            }
        }
        return temp_max; //повернутися в точку виклику search_Max(z, size);
    }

//_____

void search_Max2( int *z, int S, int * pMax, int * pIndex) // pMax- зберігає адресу змінної Max2, pIndex- зберігає адресу
змінної index
{
    int temp_max;
    int temp_index;
    int j;

    temp_max = z[0];
    temp_index = 0;

    for (j = 1; j <= S - 1; j++) {
        if (z[j] > temp_max) {
            temp_max = z[j];
            temp_index = j;
        }
    }
    *pMax = temp_max; // вміст змінної Max2
    *pIndex = temp_index; //вміст змінної index
}

//_____

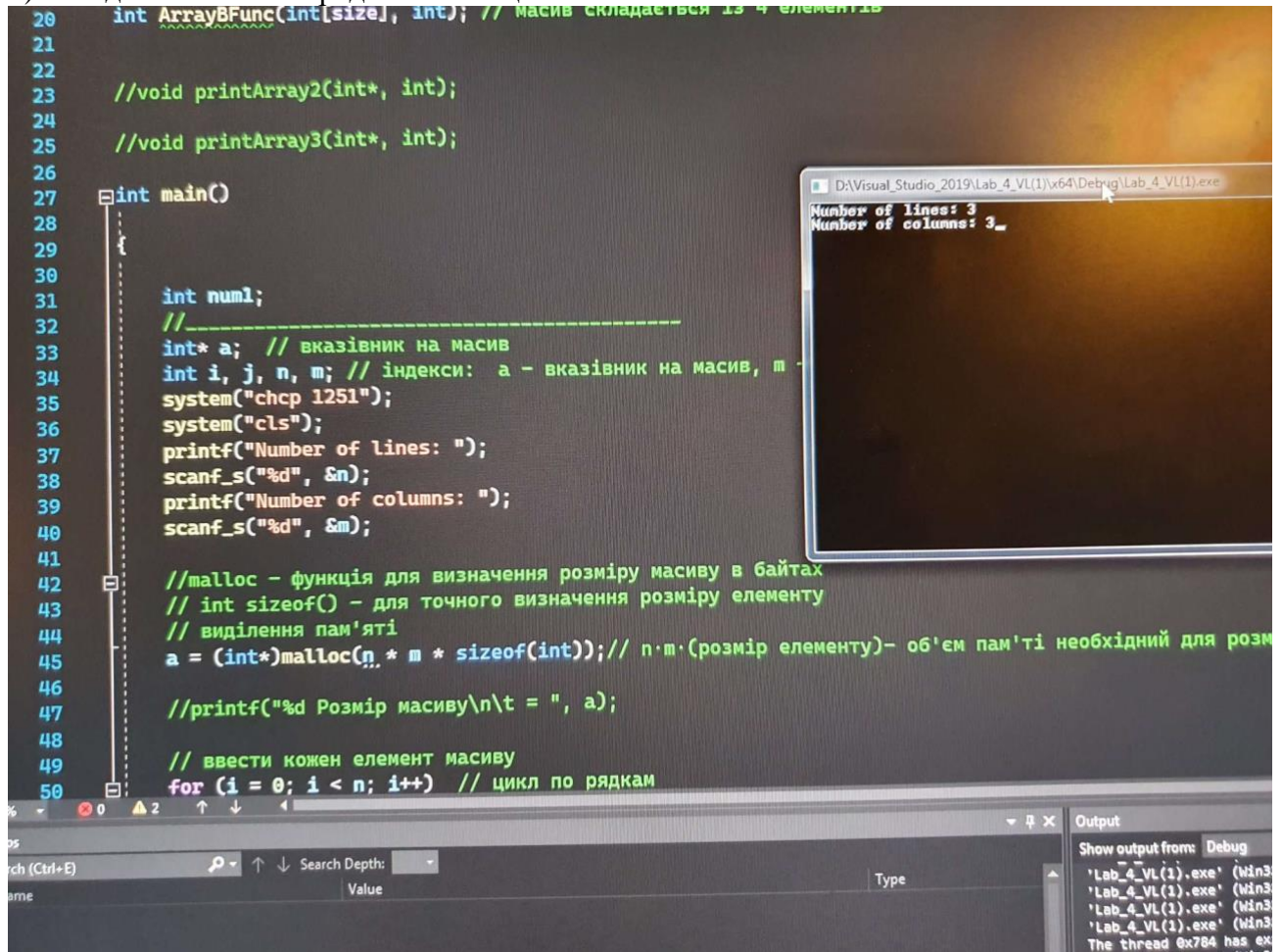
void sort_1D(int *z, int S) // сортування масиву за зростанням
{
    int pass;
    int j;
    int temp;

    for (pass = 1; pass <= S - 1; pass++) { // змінна pass пройде значення 1, 2, 3
        for (j = 0; j <= S - 1; j++) {
            if (z[j] < z[j + 1]) { // елемент j буде більший за сусідній елемент j+1, тоді в цьому випадку
міняємо їх місцями
                temp = z[j];
                z[j] = z[j + 1];
                z[j + 1] = temp;
            }
        }
    }
}

//_____

```

1) Введення кількості рядків і стовпців



```
20 int Array8Func(int[size], int); // масив складається із 4 елементів
21
22
23 //void printArray2(int*, int);
24
25 //void printArray3(int*, int);
26
27 int main()
28 {
29
30     int num1;
31     //-----
32     int* a; // вказівник на масив
33     int i, j, n, m; // індекси: a - вказівник на масив, m -
34     system("chcp 1251");
35     system("cls");
36     printf("Number of lines: ");
37     scanf_s("%d", &n);
38     printf("Number of columns: ");
39     scanf_s("%d", &m);
40
41
42     //malloc - функція для визначення розміру масиву в байтах
43     // int sizeof() - для точного визначення розміру елементу
44     // виділення пам'яті
45     a = (int*)malloc(n * m * sizeof(int)); // n*m*(розмір елементу)- об'єм пам'яті необхідний для розміру
46
47     //printf("%d Розмір масиву\n\t = ", a);
48
49     // ввести кожен елемент масиву
50     for (i = 0; i < n; i++) // цикл по рядкам
```

Number of lines: 3  
Number of columns: 3\_

Output

Show output from: Debug

'Lab\_4\_VL(1).exe' (Win32)  
'Lab\_4\_VL(1).exe' (Win32)  
'Lab\_4\_VL(1).exe' (Win32)  
'Lab\_4\_VL(1).exe' (Win32)  
The thread 0x784 has ex

2) Виконання циклу побудови матриці A

```
scanf_s("%d", &m);

//malloc - функція для визначення розміру масиву в байтах
// int sizeof() - для точного визначення розміру елементу
// виділення пам'яті
a = (int*)malloc(n * m * sizeof(int)); // n*m (розмір елементу) - об'єм пам'яті необхідний для розміщення двовимірної масиви

//printf("%d Розмір масиву\n\t = ", a);

// ввести кожен елемент масиву
for (i = 0; i < n; i++) // цикл по рядкам
{
    for (j = 0; j < m; j++) // цикл по стовпцям
    {
        printf("a[%d][%d] = ", i, j); // index = i*m+j;
        scanf_s("%d", (a + i * m + j)); // a - вказівник на масив, m - кількість стовпців
    }
}

// вивести кожен елемент масиву
for (i = 0; i < n; i++) // цикл по рядкам
{
    for (j = 0; j < m; j++) // цикл по стовпцям
    {
        //*(a + i * m + j) звернення до елементу index = i*m+j; // кожен елемент
        printf("%5d ", *(a + i * m + j)); // поле шириною 5 символів під елемент масиву
    }
    printf("\n");
}

int ArrayA = *(a + i * m + j);

num1 = *(a + 0 * m + 1); // змінюючи i та j можна викликати будь-який елемент масиву

printf("\nTransposed matrix A\n");
```

Output

```
Number of lines: 3
Number of columns: 3
a[0][0] = 3
a[0][1] = 4
a[0][2] = 5
a[1][0] = 6
a[1][1] = 7
a[1][2] = 8
a[2][0] = 9
a[2][1] = 10
a[2][2] = 11
```

3) Транспонування матриці A

```
Lab_4_VL(1) (global scope)
54 printf("a[%d][%d] = ", i, j); //index = i*m+j;
55 scanf_s("%d", (a + i * m + j)); // a - вказівник на масив, m - кількість стовпців i - індекс рядка
56 }
57 }
58 // вивести кожен елемент масиву
59 for (i = 0; i < n; i++) // цикл по рядкам
60 {
61     for (j = 0; j < m; j++) // цикл по стовпцям
62     {
63         //*(a + i * m + j) звернення до елемента index = i*m+j; // кожен елмен
64         printf("%5d ", *(a + i * m + j)); // поле шириною 5 символів під елемент масиву
65     }
66     printf("\n");
67 }
68
69 int ArrayA = *(a + i * m + j);
70
71
72 num1 = *(a + 0 * m + 1); // змінюючи i та j можна викликати будь який елемент масиву
73
74 printf("\nTransposed matrix A\n");
75 for (i = 0; i < n; i++) // ідентична частина, але результат виводиться в консоль
76 {
77     for (j = 0; j < m; j++)
78     {
79         int ArrayAT = *(a + j * m + i);
80         printf("\t %d ", ArrayAT);
81     }
82     printf("\n");
83 }
84
85 printf("\t\n");
86
87
88
```

Number of lines: 3  
Number of columns: 3  
a[0][0] = 3  
a[0][1] = 4  
a[0][2] = 5  
a[1][0] = 6  
a[1][1] = 2  
a[1][2] = 4  
a[2][0] = 5  
a[2][1] = 3  
a[2][2] = 4  
3 4 5  
6 2 4  
5 3 4

Transposed matrix A

b[0][0] =

131 % 0 2 ↑ ↓

Autos

Search (Ctrl+E) Search Depth: -

Name Value Type

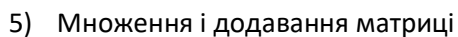
Output

Show output from: Debug

'Lab\_4\_VL(1).exe' (Win32): Loaded 'C:\Wi  
'Lab\_4\_VL(1).exe' (Win32): Loaded 'C:\Wi  
'Lab\_4\_VL(1).exe' (Win32): Loaded 'C:\Wi  
'Lab\_4\_VL(1).exe' (Win32): Loaded 'C:\Wi

4) Виконання циклу побудови матриці B





```
137 //Множення матриць
138 //ArrayA (ArrayB);
139
140 int* c;
141 c = (int*)malloc(n * m * sizeof(int));
142 int k;
143 for (i = 0; i < n; i++) // ідентична частина, але результат виводиться в консоль
144 {
145     for (j = 0; j < m; j++)
146     {
147         for (k = 0; k < m; k++) // кількість стовпців і рядків однакова
148         {
149             *c + i * m + j = 0;
150             /*int tmp2 = (*b + k * m + j);
151             int tmp3 = (*a + i * m + k);*/
152
153             *c + i * m + j += (*a + k * m + i) * (*b + k * m + j);
154             /*int nk = (*a + k * m + i) + (*b + k * m + j);
155             //printf("\nnd - [%d][%d]", nk, i, j);
156             // printf("\nnd - [%d][%d]", *c + i * m + j, i, j);
157         }
158     }
159
160 printf("\nmultiplication Matrix Result A * B ( A transposed )\n");
161 for (i = 0; i < n; i++)
162 {
163     for (j = 0; j < m; j++)
164     {
165         printf("%5dc ", *c + i * m + j);
166     }
167     printf("\n");
168 }
169
170 /*int ArrayAB= *c + i * m + j = (*a + i * m + j) * (*b + i * m + j);
171 printf("%5dc ", ArrayAB);*/
172 //
173 // Додавання матриць
174
175 int* d;
176 d = (int*)malloc(n * m * sizeof(int));
177
178 printf("\n sum of matrices A + B = \n");
179 for (i = 0; i < n; i++) // ідентична частина, але результат виводиться в консоль
180 {
181     for (j = 0; j < m; j++)
182     {
183         *d + i * m + j = *c + i * m + j + *b + i * m + j;
184         printf("\t %5d ", *d + i * m + j);
185     }
186     printf("\n");
187 }
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Microsoft Visual Studio Debug Console

```
b[0][1] = 8
b[0][2] = 2
b[1][0] = 6
b[1][1] = 8
b[1][2] = 7
b[2][0] = 2
b[2][1] = 3
b[2][2] = 4

matrix B
5c 8 2
6 8 7
2 3 4

multiplication Matrix Result L * T( A transposed )
6c 9c 12c
8c 12c 16c

sum of matrices L + T =
89 18 9
7 17 12
6 6 8
```

Output from: Debug

```
2. VL(1).exe (Win32): Loaded 'C:\Windows\System32\api-ms-win-core-timezone-l1-1-0.dll'.
2. VL(1).exe (Win32): Loaded 'C:\Windows\System32\api-ms-win-core-file-l2-1-0.dll'.
2. VL(1).exe (Win32): Loaded 'C:\Windows\System32\api-ms-win-core-synch-l1-2-0.dll'.
thread 0x40c4 has exited with code 0 (0x0).
2. VL(1).exe (Win32): Loaded 'C:\Windows\System32\apphelp.dll'.
```

6) Сортвання матриці за зростанням

```
34 //
35
36
37
38
39
40 //блок сортування матриць за зростанням
41 int sort;
42
43 printf("\n\tMatrix sorting A za zrostaniam ");
44 sort = *(a + j * m + i);
45 for (int k = 0; k < n * m; ++k) {
46     for (int i = 0; i < n; ++i) {
47         for (int j = 0; j < m; ++j) {
48             if (j != n - 1) {
49                 if (*(a + (j + 1) * m + i) < *(a + j * m + i)) {
50                     int tmp = *(a + (j + 1) * m + i);
51                     *(a + (j + 1) * m + i) = *(a + j * m + i);
52                     *(a + j * m + i) = tmp;
53                 }
54             }
55             else {
56                 if (*(a + 0 * m + (i + 1)) < *(a + j * m + i) && (i != n
57                     int tmp = *(a + 0 * m + (i + 1));
58                     *(a + 0 * m + (i + 1)) = *(a + j * m + i);
59                     *(a + j * m + i) = tmp;
60                 }
61             }
62         }
63     }
64 }
65
66 for (int i = 0; i < n; ++i) {
67     for (int j = 0; j < m; ++j)
68         printf("\t%d", *(a + j * m + i));
69 }
```

Microsoft Visual Studio Debug Console

```
13 85 9
14 6 8

The maximum element of the matrix t - 6
smallest element of the matrix t - 2
Matrix sorting t za zrostaniam 2 3 3
4 5
24 12 4 1 30 12 23 4
temp_max=30
max=30
temp_max=12
temp_max=4
temp_max=1
min=1
Max2=30 index=4
```

Output from: Debug

```
VL(1).exe* (Min32): Loaded 'C:\Windows\System32\api-ms-win-core-timezone-l1-1-0.dll'.
VL(1).exe* (Min32): Loaded 'C:\Windows\System32\api-ms-win-core-file-l2-1-0.dll'.
VL(1).exe* (Min32): Loaded 'C:\Windows\System32\api-ms-win-core-synch-l1-2-0.dll'.
end @x704 has exited with code 0 (0x0).
VL(1).exe* (Min32): Loaded 'C:\Windows\System32\api-ms-win-core-synch-l1-2-0.dll'.
```