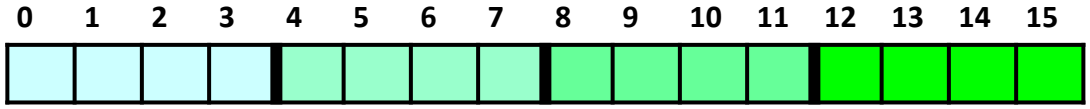
Лекция 12

Решение задач обработки матриц

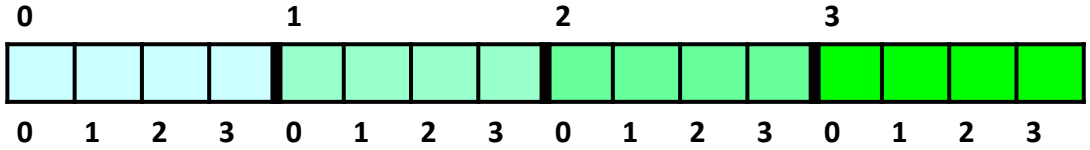
Преподаватель Палехова Ольга Александровна,
кафедра О7 БГТУ «Военмех»

Программирование матриц

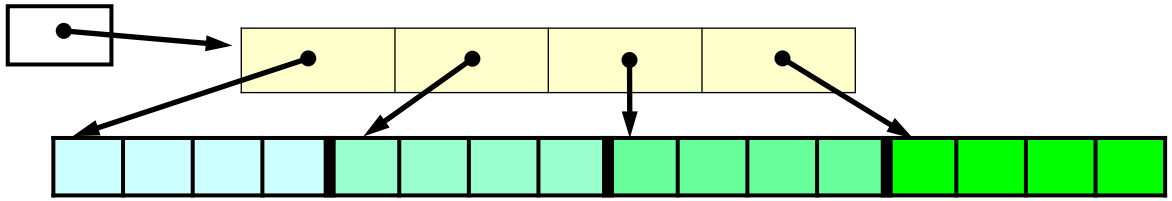
Одномерным массивом



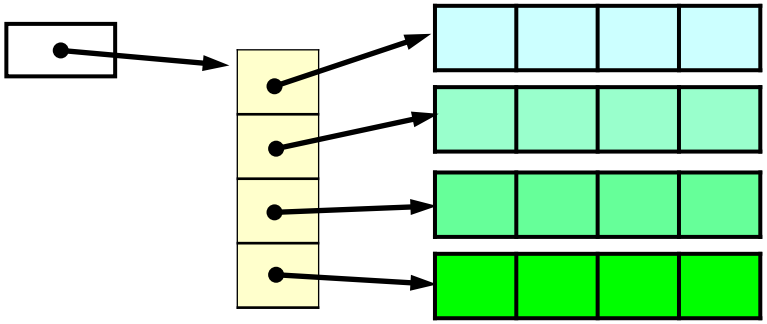
Двумерным статическим массивом



Двумерным динамическим массивом



Динамической матрицей



Эквивалентные выражения

Одномерный массив

`a`	`&a[0]`
`a+i`	`&a[i]`
`*a`	`a[0]`
`*(a+i)`	`a[i]`

В статическом двумерном массиве значения **m** и ***m** совпадают, но типы различны. В динамическом двумерном массиве и динамической матрице различны и типы, и значения этих выражений.

Двумерный массив, динамическая матрица

`m`	`&m[0]`		
`m+i`	`&m[i]`		
`*m`	`m[0]`	`&m[0][0]`	
`*(m+i)`	`m[i]`	`&m[i][0]`	
`*m+j`	`m[0]+j`	`&m[0][j]`	
`*(m+i)+j`	`m[i]+j`	`&m[i][i]`	
`**m`	`*m[0]`	`m[0][0]`	
`**(m+i)`	`*m[i]`	`m[i][0]`	
`*(*m+j)`	`*(m[0]+j)`	`(*m)[j]`	`m[0][j]`
`*(*(m+i)+j)`	`*(m[i]+j)`	`(*m+i)[j]`	`m[i][j]`

Операции с матрицами

Задача 1. Вычислить $F = \frac{S_п + S_о}{S_п - S_о}$, где $S_п$ – сумма положительных элементов в нечетных строках матрицы Y(9x12), а $S_о$ – сумма отрицательных элементов в четных строках той же матрицы.

Входные данные: матрица ***y*** размера 9x12, тип элементов не указан, для универсальности возьмем *double*.

Выходные данные: значение функции ***f*** типа *double*.

Вспомогательные переменные: ***i***, ***j*** – индексы элементов, тип *int*;
sum_negative – сумма отрицательных элементов, тип *double*;
sum_positive – сумма положительных элементов, тип *double*.

Способ выделения памяти – статический массив.

Четные строки: 2-я, 4-я, 6-я,- имеют индексы 1, 3, 5.

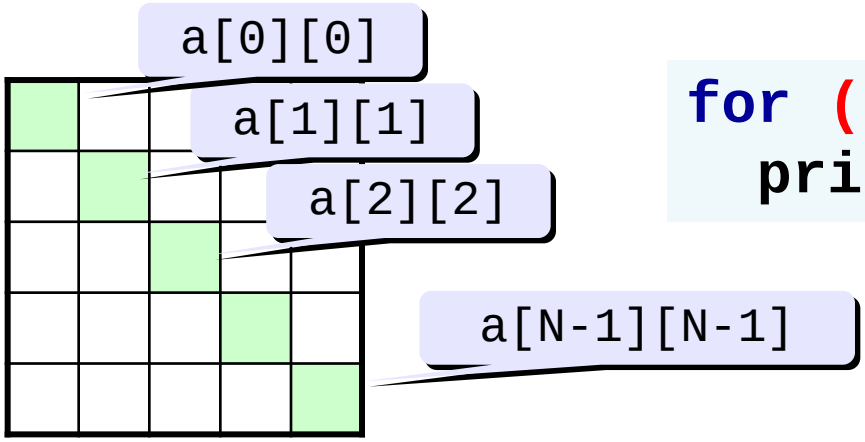
Операции с матрицами

```
#include <stdio.h>
#define M 9
#define N 12
int main()
{
    double y[M][N], sum_negative = 0, sum_positive = 0, f;
    int i, j;
    printf ("Заполните матрицу %dx%d\n", M, N);
    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            scanf ("%lf", &y[i][j]);
    for ( i = 0; i < M; i++ )
        if ( i&1 )
            for ( j = 0; j < N; j++ )
                sum_negative += y[i][j] < 0 ? y[i][j] : 0;
        else
            for ( j = 0; j < N; j++ )
                sum_positive += y[i][j] > 0 ? y[i][j] : 0;
    if ( sum_positive - sum_negative )
    {
        f = ( sum_positive + sum_negative ) / ( sum_positive - sum_negative );
        printf ("f = %f\n", f);
    }
    else
        printf ("Деление на 0\n");
    return 0;
}
```

если индекс нечетный
(номер при этом четный)

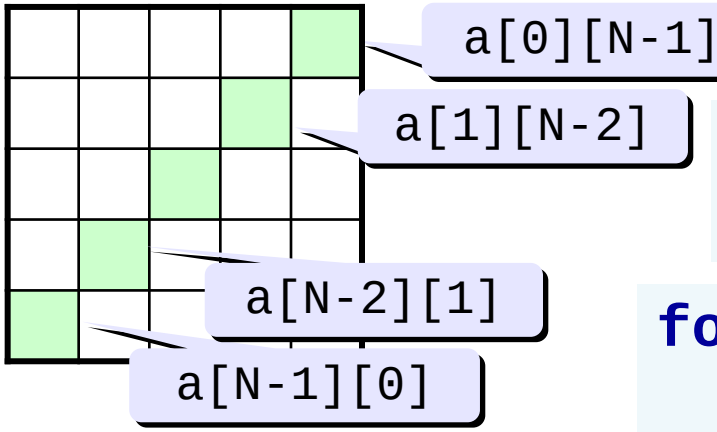
Диагонали квадратной матрицы

Задача 2. Вывести на экран элементы главной диагонали квадратной матрицы из N строк и N столбцов.




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

Задача 3. Вывести на экран элементы побочной диагонали.

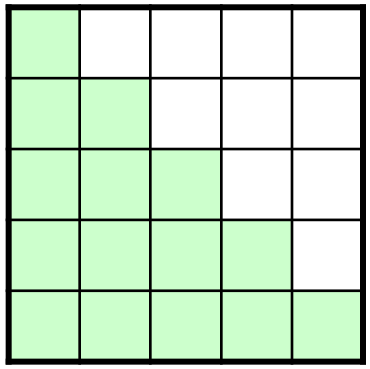



```
for (i = 0; i < N; i++)
  printf("%5d", a[i][N-1-i]);
```

```
for (i = 0; i < N; i++)
  printf("%5d", a[N-1-i][i]);
```

Операции с квадратными матрицами

Задача 4. Найти сумму элементов, стоящих на главной диагонали и под ней.



строка 0: `a[0][0]`

строка 1: `a[1][0]+ a[1][1]`

`...`

строка i: `a[i][0]+ a[i][2]+...+ a[i][i]`

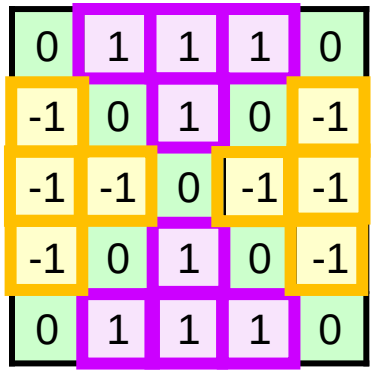
цикл по всем строкам

```
S = 0;
for ( i = 0; i < N; i++ )
  for ( j = 0; j <= i ; j++ )
    S += a[i][j];
```

только по диагональный элемент

Операции с квадратными матрицами

Задача 5. Заполнить квадратную матрицу порядка N следующим образом: на диагоналях 0, в треугольниках над и под диагоналями 1, в треугольниках слева и справа от диагоналей -1.



Верхний треугольник: в каждой **строке** элементы от главной диагонали до побочной.

Нижний треугольник симметричен верхнему, элементы обоих треугольников можно обрабатывать в одном цикле.

Левый треугольник: в каждом **столбце** элементы от главной диагонали до побочной

Правый треугольник симметричен левому, элементы обоих треугольников можно обрабатывать в одном цикле.

Заполнение квадратной матрицы

0	1	1	1	0
-1	0	1	0	-1
-1	-1	0	-1	-1
-1	0	1	0	-1
0	1	1	1	0

Диагонали:

```
for ( i = 0; i < N; i++ )
  a[i][i] = a[i][N-1-i] = 0;
```

Треугольники:

цикл по половине строк и столбцов (верхний и левый треугольники)

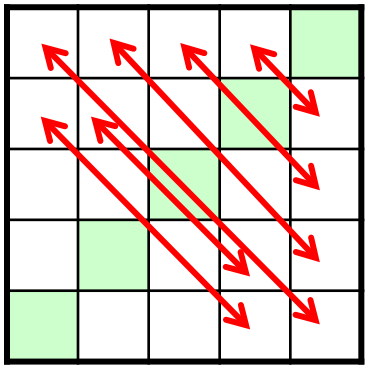
от главной диагонали до побочной

```
for ( i = 0; i < N / 2; i++ )
  for ( j = i+1; j < N-1-i; j++ )
  {
    a[i][j] = a[N-1-i][j] = 1;
    a[j][i] = a[j][N-1-i] = -1;
  }
```

верхний и нижний треугольники

левый и правый треугольники

Симметрия квадратной матрицы



Задача 6. Определить, является ли заданная целочисленная квадратная матрица 9x9 симметричной относительно побочной диагонали.

Алгоритм:

- устанавливаем ***флаг***
- в цикле по всем элементам одного треугольника
  - сравниваем a_{ij} и $a_{N-j\ N-i}$
  - если $a_{ij} \neq a_{N-j\ N-i}$ сбрасываем флаг
- если ***флаг*** установлен, матрица симметрична

Симметрия квадратной матрицы

1 способ организации матрицы – *статический двумерный массив*.

```
#include <stdio.h>
#define N 9
int main()
{
    int a[N][N];
    int i, j, flag = 1;
    /* ввод матрицы */
    for ( i = 0; i < N-1 && flag; i++ )
        for ( j = 0; j < N-1-i ; j++ )
            if ( a[i][j] != a[N-1-j][N-1-i] )
            {
                flag = 0;
                break;
            }
    printf ("Матрица %sсимметрична\n", flag?"":"не ");
    return 0;
}
```

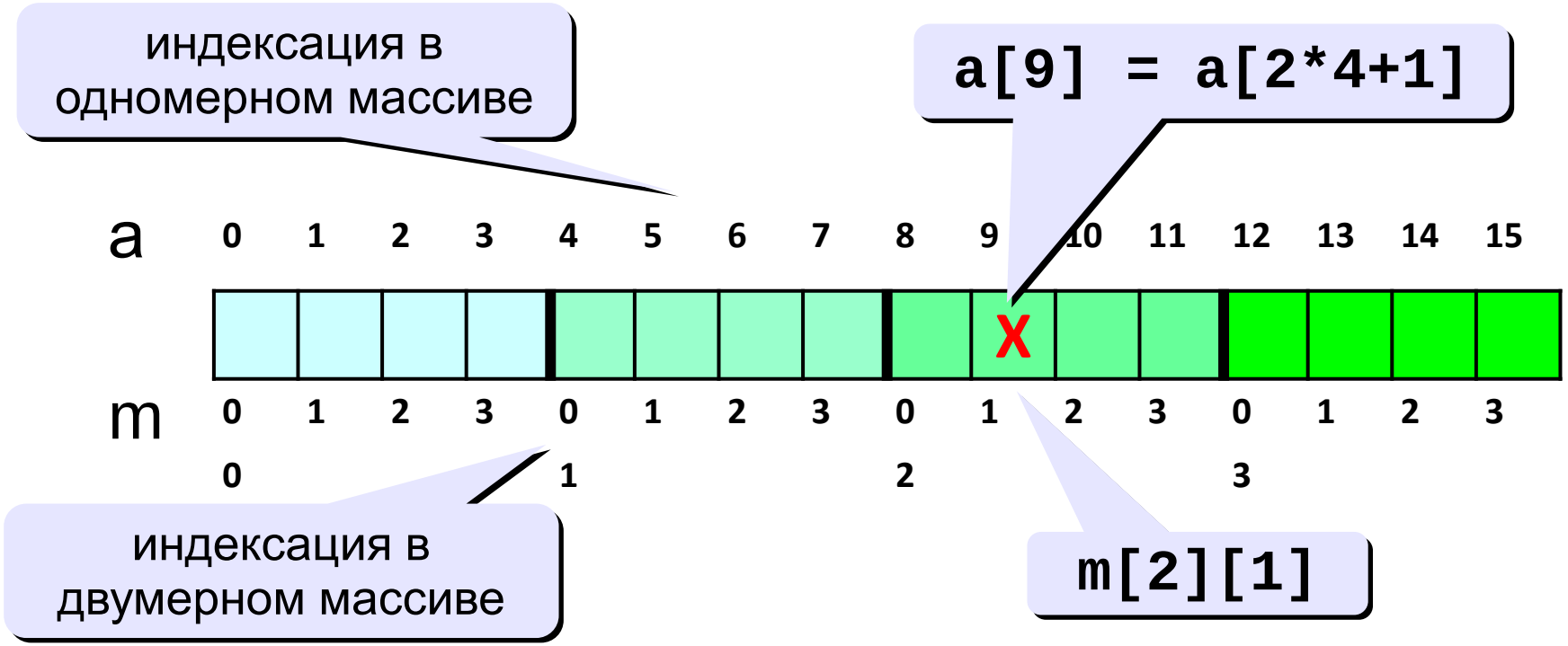
над диагональю N-1 строка

от 0 до диагонали

элементы, расположенные симметрично относительно побочной диагонали

Программирование матрицы одномерным массивом

2 способ организации матрицы – *статический одномерный массив*.



! Для матрицы с N столбцами

`m[i][j] → a[i*N+j]`

Программирование матрицы одномерным массивом

2 способ организации матрицы – *статический одномерный массив*.

```
#include <stdio.h>
#define N 9
int main()
{
    int a[N*N];
    int i, j, flag = 1;
    for ( i = 0; i < N*N; i++ )
        scanf ("%d", &a[i]);
    for ( i = 0; i < N-1 && flag; i++ )
        for ( j = 0; j < N-1-i ; j++ )
            if ( a[i*N+j] != a[ (N-1-j) * N + N-1-i ] )
            {
                flag = 0;
                break;
            }
    printf ("Матрица %sсимметрична\n", flag?"":"не ");
    return 0;
}
```

объявление массива

заполняем весь массив

вычисляем смещение на i строк и j элементов в строке

Операции с матрицами

Задача 7. Дана целочисленная квадратная матрица 7-го порядка, элементы которой уникальны. Найти и поменять местами максимальные среди элементов матрицы, лежащих над и под главной диагональю.

Входные данные: матрица ***a*** размера 7x7, тип элементов *int*.

Выходные данные: та же матрица с измененными значениями.

Замечание: чтобы поменять элементы, нужно знать их местоположение, для чего можно использовать или индексы, или адрес. При использовании Си предпочтительнее адрес.

Вспомогательные переменные: ***i***, ***j*** – индексы элементов, тип *int*;
max1 – адрес максимального элемента над диагональю, тип *int**;
max2 – адрес максимального элемента под диагональю, тип *int**.

Способ выделения памяти – статический двумерный массив.

Операции с матрицами

```
#include <stdio.h>
#define N 7
int main()
{
    int a[N][N], *max1, *max2, tmp;
    int i, j;
    /* ввод матрицы */
    max1 = &a[0][1]; max2 = &a[1][0];
    for ( i = 1; i < N; i++ )
        for ( j = 0; j < i; j++)
        {
            if ( a[i][j] > *max1 ) max1 = &a[i][j];
            if ( a[j][i] > *max2 ) max2 = &a[j][i];
        }
    tmp = *max1;
    *max1 = *max2;
    *max2 = tmp;
    /* вывод матрицы */
    return 0;
}
```

до главной диагонали

над диагональю

под диагональю

Сортировка столбцов матрицы

Задача 8. Отсортировать элементы каждого столбца заданной прямоугольной матрицы в порядке убывания.

Входные данные: количество строк и столбцов матрицы, ***m*** и ***n*** типа *int,*
матрица ***a*** размера ***m***x***n***, тип элементов не указан, для универсальности возьмем *double*.

Выходные данные: та же матрица после упорядочения.

Вспомогательные переменные: ***i***, ***j, k*** – индексы элементов, тип *int*;
tmp – вспомогательная переменная для обмена элементов, тип *double.*

Способ выделения памяти – динамический двумерный массив.

Сортировка столбцов матрицы

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    double **a, tmp;
    int m, n, i, j, k;
    printf ("Количество строк матрицы = ");
    scanf ("%d", &m);
    printf ("Количество столбцов матрицы = ");
    scanf ("%d", &n);
    a = calloc (m, sizeof(double*));
    if ( a == NULL )
    {
        printf ("Ошибка выделения памяти\n");
        return 100;
    }
    a[0] = calloc (m*n, sizeof(double));
    if ( a[0] == NULL )
    {
        free (a);
        printf ("Ошибка выделения памяти\n");
        return 100;
    }
    for ( i = 1; i < m; i++ )
        a[i] = a[i-1] + n;
```

выделение памяти под массив указателей на строки

проверка выделения памяти

выделение памяти под элементы матрицы

если память под данные не выделена, освобождаем блок указателей

расстановка указателей

Сортировка столбцов матрицы

```
    printf ("Заполните матрицу %dx%d\n", m, n);
    for ( i = 0; i < m; i++ )
        for ( j = 0; j < n; j++ )
            scanf ("%lf", &a[i][j]);
    for ( k = 0; k < n; k++ )
        for ( i = 1; i < m; i++ )
            for ( j = 0; j < m-i; j++)
                if ( a[j][k] < a[j+1][k] )
                {
                    tmp = a[j][k];
                    a[j][k] = a[j+1][k];
                    a[j+1][k] = tmp;
                }
    for ( i = 0; i < m; i++ )
    {
        for ( j = 0; j < n; j++ )
            printf ("%8.2f", a[i][j]);
        printf ("\n");
    }
    free (a[0]);
    free (a);
    return 0;
}
```

цикл по столбцам

цикл по элементам столбца

сортировка одного столбца «пузырьком»

освобождение памяти

Перестановка строк матрицы

Задача 9. Переставить строки заданной матрицы размера 7x8 таким образом, чтобы элементы в первом столбце оказались упорядоченными по возрастанию.

Входные данные: матрица ***a*** размера 7x8, тип элементов не указан, для универсальности возьмем *double*.

Выходные данные: та же матрица после упорядочения.

Вспомогательные переменные: ***i***, ***j, k*** – индексы элементов, тип *int*;
tmp – вспомогательная переменная для обмена элементов, тип *double.*

1 способ выделения памяти – статический двумерный массив.

Перестановка строк матрицы

```
#include <stdio.h>
#define M 7
#define N 8
int main()
{
    double a[M][N], tmp;
    int i, j, k;
    /* ввод матрицы */
    for ( i = 1; i < M; i++ )
        for ( j = 0; j < M-i; j++)
            if ( a[j][0] > a[j+1][0] )
                for ( k = 0; k < N; k++ )
                {
                    tmp = a[j][k];
                    a[j][k] = a[j+1][k];
                    a[j+1][k] = tmp;
                }
    /* вывод матрицы */
    return 0;
}
```

Для обмена строк нужно много действий

сравниваем первые элементы строк

сортировка «пузырьком»

обмен строк

Перестановка строк матрицы

2 способ выделения памяти – динамическая матрица.

Вспомогательные переменные: ***i***, ***j, k*** – индексы элементов, тип *int*;
tmp – вспомогательная переменная для обмена строк, тип *double*.*

```
#include <stdio.h>
#include <stdlib.h>
#define M 7
#define N 8
int main()
{
    double **a, *tmp;
    int i, j;
    a = calloc (M, sizeof(double*));
    if ( a == NULL )
    {
        printf ("Ошибка выделения памяти\n");
        return 100;
    }
    for ( i = 0; i < M; i++ )
    {
        a[i] = calloc (N, sizeof(double));
        if ( a[i] == NULL )
        {
            for ( --i; i >= 0; i-- )
                free (a[i]);
            free (a);
            printf ("Ошибка выделения памяти\n");
            return 100;
        }
    }
```

выделение памяти под массив указателей на строки

если память под часть данных не выделена, освобождаем уже имеющиеся блоки данных и блок указателей

Перестановка строк матрицы

```
    printf ("Заполните матрицу %dx%d\n", M, N);
    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            scanf ("%lf", &a[i][j]);
    for ( i = 1; i < M; i++ )
        for ( j = 0; j < M-i; j++)
            if ( a[j][0] > a[j+1][0] )
            {
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
    for ( i = 0; i < M; i++ )
    {
        for ( j = 0; j < N; j++ )
            printf ("%8.2f", a[i][j]);
        printf ("\n");
    }
    for ( i = 0; i < M; i++ )
        free (a[i]);
    free (a);
    return 0;
}
```

сравниваем первые элементы строк

сортировка «пузырьком»

обмен строк

освобождение памяти

!

Другой способ организации матрицы позволил работать быстрее

Удаление столбцов матрицы

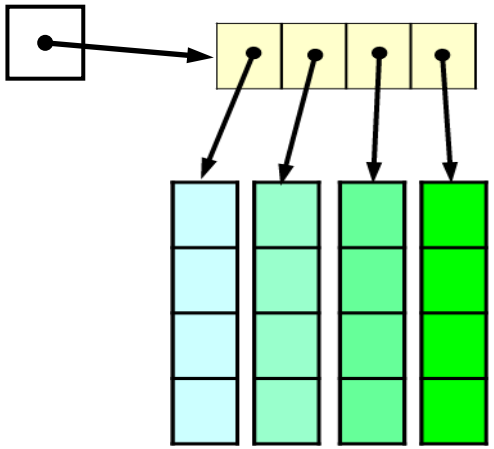
Задача 10. Удалить из заданной целочисленной матрицы столбцы, содержащие нули.

Входные данные: количество строк и столбцов матрицы, ***m*** и ***n*** типа *int,*
матрица ***a*** размера ***m***x***n***, тип *int*.

Выходные данные: та же матрица после удаления столбцов, количество столбцов.

Вспомогательные переменные: ***i***, ***j, k*** – индексы элементов, тип *int.*

Способ выделения памяти – динамическая матрица с выделением памяти под столбцы.



Удаление столбцов матрицы

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int **a;
    int m, n, i, j, k;
    /* ввод количества строк и столбцов */
    a = calloc (n, sizeof(int*));
    if ( a == NULL )
    {
        printf ("Ошибка выделения памяти\n");
        return 100;
    }
    for ( i = 0; i < n; i++ )
    {
        a[i] = calloc (m, sizeof(int));
        if ( a[i] == NULL )
        {
            for ( --i; i >= 0; i-- )
                free (a[i]);
            free (a);
            printf ("Ошибка выделения памяти\n");
            return 100;
        }
    }
```

массив указателей по числу столбцов

число элементов в каждом столбце равно числу строк

Удаление столбцов матрицы

```
printf ("Заполните матрицу %dx%d\n", m, n);
for ( i = 0; i < m; i++ )
    for ( j = 0; j < n; j++ )
        scanf ("%d", &a[j][i]);
for ( j = 0, k = 0; j < n; j++ )
{
    for ( i = 0; i < m && a[j][i]; i++);
        if ( i < m )
        {
            free (a[j]);
            a[j] = NULL;
        }
        else
            a[k++] = a[j];
}
```

заполняем матрицу по строкам, внешний цикл по второму индексу

цикл по столбцам

ищем нуль в столбце

если нуль есть

освобождаем память и обнуляем указатель

число оставшихся столбцов равно ***k***

если нуля нет, то «перевешиваем» столбец на новое место

Удаление столбцов матрицы

```
    n = k;
    if ( n )
        for ( i = 0; i < m; i++ )
        {
            for ( j = 0; j < n; j++ )
                printf ("%8d", a[j][i]);
            printf ("\n");
        }
    else
        printf ("Матрица пуста\n");
    for ( i = 0; i < n; i++ )
        free (a[i]);
    free (a);
    return 0;
}
```

новое число столбцов

вывод матрицы по строкам

освобождение памяти