Глава 1

Примеры решения задач с использованием массивов

1.1. Примеры работы с одномерными массивами

Работа с одномерными массивами в итоге сводится к нескольким общим задачам:

1. Поиск суммы (произведения, разности и т. д.) элементов, попадающих (не попадающих) в заданный интервал.

2. Поиск минимума (максимума) элементов массива.

3. Замена значений элементов, удовлетворяющих некоторому условию.

   Если надо заменить значение элемента, удовлетворяющего условию, например, нулем, то оператор выглядит следующим образом: `a[i] = 0`.

   Если нужно поменять местами значения двух элементов, то можно воспользоваться встроенной функцией `swap`: `swap(a[i], a[j])`.

Во всех случаях рассматриваются именно ЗНАЧЕНИЯ элементов. Если необходимо работать с индексами элементов, в условии задачи будет особо выделено, что речь идет именно об индексах (Например, найти сумму элементов с четными порядковыми номерами.)

Поиск элементов, попадающих (не попадающих) в заданный интервал

Условие поиска элементов, попадающих (не попадающих) в заданный интервал `[a,b]`: необходимо определить удовлетворяет ли ЗНАЧЕНИЕ ЭЛЕМЕНТА массива следующим неравенствам:

- Элемент массива попадает в заданный интервал:

$$a \leqslant mas[i] \leqslant b.$$

  Условие, записанное на языке C++:

```
if (mas[i] >= a && mas[i] <= b)...
```

- Элемент массива не попадает в заданный интервал:

$$mas[i] \leqslant a \vee mas[i] \geqslant b.$$

Условие, записанное на языке C++:

```
if (mas[i] < a || mas[i] > b)...
```

Пример 1.1. Дан массив, содержащий целые числа. Найти среднее арифметическое элементов, кратных трем, не попадающих в заданный интервал `[a,b]`. Если таких чисел нет, вывести сообщение об этом.

Например,

n	**Массив**	**Диапазон**	**Результат**
4	9, 4, 6, 3	[4, 7]	6
4	8, 4, 6, 2	[4, 7]	таких элементов нет

Листинг 1.1. Код программы

```
#include<iostream>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
using namespace std;

int main(){
  setlocale (LC_ALL,"rus");                    //вывод данных русском языке
  int n;
  cout << "n="; cin >> n;                      //размер массива
  int *a = new int [n];                        //выделение памяти под массив
  srand ((unsigned)time(NULL));                // начальная точка генерации
  for (int i = 0;  i < n; i++){
    a[i] = rand() % 15;                       //псевдослучайное число
    cout << a[i] << " ";
  }
  cout << endl;
  int b,c;
  cout << "b="; cin >> b;                      //ввод диапазона
  cout << "c="; cin >> c;
  if (b > c) swap(b,c);                        //замена значений, если b > c
  int k = 0;
```

```
23   int sum = 0;
24   for (int i = 0; i < n; i++)
25     if (a[i]%3 == 0 && ( a[i] < b || a[i] > c)){ //поиск элементов
26       k++;                                      //количество элементов
27       sum += a[i];                              //сумма значений элементов
28     }
29   if (!k) cout << "Таких элементов нет\n"; //k = 0
30   else
31     cout << "Среднее арифметическое: " << 1.0*sum/k <<endl;
32   return 0;
33 }
```

Рассмотрим работу программы 1.1 для первого примера:

i	$a[i]$	строка 25 листинга 1.1	k	S
н.у.			0	0
0	9	`true`	1	9
1	4	`false` (попадает в интервал)	1	9
2	6	`false` (кратно трем, но попадает в интервал)	1	9
3	3	`true`	2	12

В итоге среднее арифметическое равно 6. □

Поиск минимальных (максимальных) элементов массива

Для того, чтобы определить экстремумы массива, необходимо сравнивать каждый элемент с текущим экстремумом, и, если значение элемента меньше (больше) текущего экстремума, присвоить текущему экстремума значение этого элемента. Т. е.:

```
if (mas[i] < min) min = mas[i];
```

Для того, чтобы начать сравнение, необходимо присвоить начальное значение переменной `min`. Возможны два варианта:

- Если надо найти минимум или максимум в массиве, то переменной `min` присваивается значение первого элемента массива, т. е., `min = a[0]`.

  Поскольку в массиве обязательно существует и минимальное и максимальное значение элементов (в случае, если массив состоит из одинаковых элементов, то его минимум и максимум совпадают), такое присваивание позволяет гарантировано определить искомые экстремумы. Например, фрагмент кода программы 1.2 позволяет найти минимальное и максимальное значение в массиве:

Листинг 1.2. Поиск экстремумов массива

```
1 ...
2 int min = a[0];
3 int max = a[0];
4 for (int i = 1; i < n; i++){
5   if (a[i] < min)
6     min = a[i];
7   if (a[i] > max)
8     max = a[i];
9 }
```

Рассмотрим массив `a[4] = {1, 5, 0, 9}`:

i	$a[i]$	строка 6 листинга 1.2	min	строка 8 листинга 1.2	max
0	1		1		1
1	5	`false`	1	`true`	5
2	0	`true`	0	`false`	5
3	9	`false`	0	`true`	9

- В случае, если необходимо найти экстремум из элементов, удовлетворяющих некоторому условию (например, максимум из четных элементов), нельзя присваивать начальному значению переменной `min` (или `max`) значение первого элемента.

  Например, надо найти максимум из четных элементов массива.

  Пусть есть следующий массив: `a[4] = {9, 2, 8, 1}`. Пусть начальное значение переменной `max = a[0]`.

  Рассмотрим пошаговую реализацию следующего фрагмента кода программы:

Листинг 1.3. Поиск максимума четных элементов массива

```
1 ...
2 int max = a[0];
3 for (int i = 1; i < n; i++){
4   if (!(a[i] % 2) && a[i] > max)     //если число четное и > max
5     max = a[i];
6 }
```

i	$a[i]$	строка 4 листинга 1.3	max
0	9		9
1	2	`false` (четное, но меньше `max`)	9
2	8	`false` (четное, но меньше `max`)	9
3	1	`false` (нечетное)	9

В итоге получили неправильный ответ.

В таком случае, самый простой вариант выбирать в качестве начального значения переменных предельные значения для соответствующего типа, например, если необходимо найти максимум, то в качестве начального значения переменной `max` надо выбрать минимально возможное значение для этого типа. Тогда любое значение элементов массива было гарантировано больше начального значения переменной `max`.

Для этого необходимо подключить библиотеку `<limits.h>`, в которой хранятся предельные значения для порядковых типов данных на данном компиляторе или `<float.h>`, в которой хранятся предельные значения для порядковых типов данных на данном компиляторе.

Значения записаны во встроенных переменных, их имена можно посмотреть в справочнике. Например, для типа `int` предельные значения хранятся в переменных `INT_MIN` и `INT_MAX`. Тогда листинг 1.3 будет иметь вид:

Листинг 1.4. Поиск максимума четных элементов массива

```
1 #include<limits.h>
2 ...
3 int max = INT_MIN;
4 for (int i = 0; i < n; i++){
5   if (!(a[i] % 2) && a[i] > max)
6     max = a[i];
7 }
```

i	$a[i]$	строка 6 листинга 1.3	max
нач. усл.			-2147483648
0	9	`false` (нечетное)	-2147483648
1	2	`true`	2
2	8	`true`	8
3	1	`false` (нечетное)	8

Ответ: `max = 8.`

Пример 1.2. Дан массив, содержащий целые числа. Заменить все минимальные элементы нулем.

Листинг 1.5. Код программы

```
#include<iostream>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
using namespace std;

int main(){
  setlocale (LC_ALL,"rus");                    //вывод данных на русском языке
  int n;
  cout << "n="; cin >> n;                      //размер массива
  int *a = new int [n];                        //выделение памяти под массив
  srand ((unsigned)time(NULL));                // начальная точка генерации
  for (int i = 0;  i < n; i++){
    a[i] = rand() % 15;                       //псевдослучайное число
    cout << a[i] << " ";
  }
  cout << endl;
  int min = a[0];                              //минимум = нулевой элемент
  for (int i = 1; i < n; i++)
    if (a[i] < min) min = a[i];               //поиск минимума
  for (int i = 0; i < n; i++)
    if (a[i] == min) a[i] = 0;                //заменяем нулем минимумы
  for (int i = 0; i < n; i++)                  //вывод на экран нового массива
    cout << a[i] << " ";
  cout << endl;
  return 0;
}
```

Результаты работы программы:

$n = 10$ До	$n = 10$ После
$8, 14, 8, 7, 8, 4, 2, 2, 8, 2$	$8, 14, 8, 7, 8, 4, 0, 0, 8, 0$
$1, 3, 8, 7, 14, 4, 7, 2, 10, 1$	$0, 3, 8, 7, 14, 4, 7, 2, 10, 0$

□

Пример 1.3. Вывести номера максимальных четных элементов.

Листинг 1.6. Код программы

```
#include<iostream>
#include<math.h>
#include<limits.h>
using namespace std;

int main(){
  int n;
  cout << "n ="; cin >> n;          //ввод размерности
  int *a = new int [n];
  for (int i = 0; i < n; i++){      //заполняем массив элементами
    cout << "a[" << i << "]=";
    cin >> a[i];
  }
  int max = INT_MIN;
  for (int i = 0; i < n; i++)       //поиск максимального четного элемента
    if (!(a[i] % 2) && a[i] > max)
      max = a[i];
  if (max == INT_MIN)               //четных элементов в массиве нет
    cout << "четных элементов нет\n";
  else
    for (int i = 0; i < n; i++)    //выводим на экран номера max
      if (a[i] == max)
        cout << i << " ";
  cout << endl;
  return 0;
}
```

Результат работы программы:

Размерность	Массив	Результат
$n = 10$	$8, 9, 8, 7, 8, 4, 2, 2, 8, 2$	$0, 2, 4, 8$
$n = 10$	$7, 9, 3, 7, 5, 1, 9, 7, 7, 3$	четных элементов нет

□

1.1.1. Примеры работы с двумерными массивами

Работа с двумерными массивами также сводится к нескольким общим задачам. Некоторые задачи, такие как поиск экстремумов в массиве, поиск значений, попадающих в заданный интервал, замена значений элементов на новые, уже были рассмотрены для случая одномерных массивов и отличаются только наличием вложенного цикла (изменение как параметра `i`, так параметра `j`).

Рассмотрим более подробно задачи, связанные с поиском элементов в строке или столбце.

Поиск экстремумов в строках или столбцах

Пусть есть задача найти сумму максимальных элементов каждой строки.

Т. е., для каждой строки надо определить свой локальный максимум. Поскольку строку можно рассматривать как одномерный массив, то поиск максимума в строке был показан в разделе 1.1. Необходимо обязательно при переходе к рассмотрению новой строки переопределять значение переменной `max.`

Например, фрагмент кода программы 1.7 позволяет найти сумму максимальных элементов каждой строки:

Листинг 1.7. Поиск экстремумов массива

```
1 ...
2 int sum = 0;
3 for (int i = 0; i < n; i++){
4   int max = a[i][0];              //начальное значение для каждой строки
5   for (int j = 1; j < m; j++)
6     if (a[i][j] > max)
7       max = a[i][j];            //max для i-ой строки
8   sum += max;                     //находим сумму
9 }
```

Для `i`-ой строки массива сначала присваивается переменной `max` значение первого элемента `i`-ой строки массива (строка 4 листинга 1.7). После этого производится поиск максимального элемента `i`-ой строки массива (строки 5–7 листинга 1.7). Далее полученное значение добавляется к искомой сумме (строка 8 листинга 1.7).

Пусть есть массив $\begin{pmatrix} 4 & 5 & \mathbf{9} \\ 2 & \mathbf{4} & 1 \\ \mathbf{7} & 3 & 4 \end{pmatrix}$.

Результат работы программы для этого массива:

i	j	$a[i][j]$	max	sum
0	0	4	4	0
0	1	5	5	0
0	2	9	9	9
1	0	2	2	9
1	1	4	4	9
1	2	1	4	13
2	0	7	7	13
2	1	3	7	13
2	2	4	7	20

Ответ `sum = 20.`

Обмен строк и столбцов

Строки и столбцы можно менять местами только поэлементно. Фиксируем номера строк (столбцов) и меняем местами все элементы этих строк (столбцов).

Листинг 1.8. Поиск экстремумов массива

```
1  ...
2  for (int i = 0; i < n/2; i++){          //меняем первую строку с последней и т.д.
3    for (int j = 0; j < m; j++)
4      swap(a[i][j],a[n-1-i][j]);
5  }
6  ...
7  for (int j = 0; j < m; j += 2){         //меняем первый столбец со вторым и т.д.
8    for (int i = 0; i < n; i++)
9      swap(a[i][j],a[i][j+1]);
10 }
```

В строках 2–5 листинга 1.8 меняются местами первая и последняя строки, потом вторая и предпоследняя и т. д. Условие завершения цикла (строка 2):

`i < n/2.`

Пусть счетчик в операторе цикла будет изменятся до `n.`

Результаты такой замены представлены ниже для массива размерностью 4×3:
$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$.

В таблице полужирным шрифтом выделена `i`-ая строка, а курсивом — `(n - 1 - i)`-ая строка:

i	$n-1-i$	матрица	результат
0	3	$\begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathit{4} & \mathit{4} & \mathit{4} \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$
1	2	$\begin{pmatrix} 4 & 4 & 4 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} \\ \mathit{3} & \mathit{3} & \mathit{3} \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 3 & 3 & 3 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix}$
2	1	$\begin{pmatrix} 4 & 4 & 4 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} \\ \mathit{2} & \mathit{2} & \mathit{2} \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$
3	0	$\begin{pmatrix} \mathbf{4} & \mathbf{4} & \mathbf{4} \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathit{1} & \mathit{1} & \mathit{1} \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$

Как видно, до середины цикла строки будут меняться правильно. Далее будут меняться уже новые строки, и в итоге получится первоначальная матрица.

Поэтому в строке 2 листинга 1.8 цикл должен ОБЯЗАТЕЛЬНО продолжаться только до `n/2`.

В строках 7–10 производится замена первого столбца со вторым, третьего с четвертым и т. д. Так как меняются столбцы с четными порядковыми номерами с соседними им столбцами, счетчик цикла (строка 7 листинга 1.8) должен увеличиваться на два.

В случае, если необходимо поменять местами строки, содержащие какие-либо элементы, то можно воспользоваться следующим приемом:

1. Создать переменную типа `bool` и присвоить ей значение `true` (предполагаем, что все элементы удовлетворяют нужному условию.)

2. Проверить удовлетворяют ли нужному условию все элементы строки. Если нет, присвоить переменной типа `bool` значение `false` и завершить цикл с помощью операции `break`.

3. Если переменная типа `bool` имеет значение `true`, выполнять необходимые действия.

Фрагмент кода программы (листинг 1.9) демонстрирует поиск в двумерном массиве столбца, содержащего только двузначные элементы и водит на экран номера столбцов:

Листинг 1.9. Поиск экстремумов массива

```
1  ...
2  for (int j = 0; j < m; j++){                //для каждого столбца
3    bool flag = true;
4    for (int i = 0; i < n; i++)
5      if (a[i][j] <= 9 || a[i][j] >= 100){ //число не двузначное
6        flag = false;
7        break;                          //прерываение цикла
8      }
9    if (flag) cout << j << " ";           //если flag = true
10 }
11 ...
```

Диагонали матрицы

В случае квадратной матрицы (размерности $n \times n$) часто встречаются задачи, связанные с работой с диагоналями матрицы (например, один из способов нахождения детерминанта матрицы: преобразовать матрицу в треугольную, потом найти произведение элементов, расположенных на главной диагонали). Если матрица является прямоугольной, о диагоналях говорить нельзя.

У матрицы существует две диагонали: *главная* и *побочная*. На рисунке 1.1 представлены все случаи расположения элементов двумерного массива относительно диагоналей матрицы. Главная диагональ представлена на рисунке 1.1 a.–b., побочная — на рисунке 1.1 c.–d.

Рассмотрим массив размерностью $n \times n$. Слева полужирным шрифтом выделена главная диагональ этого массива, справа — побочная:

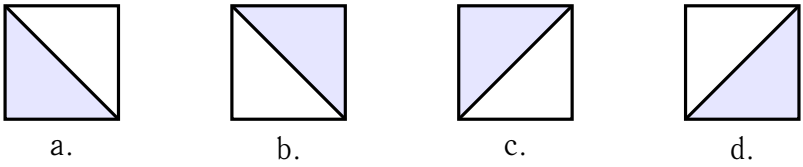


Рис. 1.1. Диагонали матрицы

$$\begin{pmatrix} \mathbf{1} & 2 & 3 & 4 & 5 \\ 1 & \mathbf{2} & 3 & 4 & 5 \\ 1 & 2 & \mathbf{3} & 4 & 5 \\ 1 & 2 & 3 & \mathbf{4} & 5 \\ 1 & 2 & 3 & 4 & \mathbf{5} \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 & 4 & \mathbf{5} \\ 1 & 2 & 3 & \mathbf{4} & 5 \\ 1 & 2 & \mathbf{3} & 4 & 5 \\ 1 & \mathbf{2} & 3 & 4 & 5 \\ \mathbf{1} & 2 & 3 & 4 & 5 \end{pmatrix}$$

Чтобы определить условие расположения элемента относительно диагоналей, выпишем индексы двумерного массива:

$$\begin{pmatrix} \mathbf{00} & 01 & 02 & 03 & 04 \\ 10 & \mathbf{11} & 12 & 13 & 14 \\ 20 & 21 & \mathbf{22} & 23 & 24 \\ 30 & 31 & 32 & \mathbf{33} & 34 \\ 40 & 41 & 42 & 43 & \mathbf{44} \end{pmatrix} \quad \begin{pmatrix} 00 & 01 & 02 & 03 & \mathbf{04} \\ 10 & 11 & 12 & \mathbf{13} & 14 \\ 20 & 21 & \mathbf{22} & 23 & 24 \\ 30 & \mathbf{31} & 32 & 33 & 34 \\ \mathbf{40} & 41 & 42 & 43 & 44 \end{pmatrix}$$

Как видно из представленных примеров, на главной диагонали расположены следующие элементы массива:

```
a[0][0], a[1][1], a[2][2], ..., a[n][n].
```

Следовательно, условие нахождения элемента на главной диагонали:

$$i = j.$$

На побочной диагонали расположены следующие элементы массива:

```
a[0][n - 1], a[1][n - 2], a[2][n - 3], ..., a[n - 1][0].
```

Следовательно, условие нахождения элемента на главной диагонали:

$$j = n - 1 - i.$$

В таблице 1.1 представлены все случаи расположения элементов двумерного массива относительно диагоналей.

Таблица 1.1. Расположение элементов относительно диагоналей матрицы

Расположение	Условие	Рисунок	Код программы

Выше главной	$j > i$	Рис. 1.1b.	`for(int i = 0; i < n; i++)`
			`  for (int j = i + 1; j < n; j++)`
			`  ...`

Ниже главной	$j < i$	Рис. 1.1a.	`for(int i = 0; i < n; i++)`
			`  for (int j = 0; j < i; j++)`
			`  ...`

Выше побочной	$j < n - 1 - i$	Рис. 1.1c.	`for(int i = 0; i < n; i++)`
			`  for (int j = 0; j < n - 1 - i; j++)`
			`  ...`

Ниже побочной	$j > n - 1 - i$	Рис. 1.1d.	`for(int i = 0; i < n; i++)`
			`  for (int j = n - 1 - i; j < n; j++)`
			`  ...`

Примеры задач

Пример 1.4. Поменять местами строки, содержащие первый минимальный и последний максимальный элементы массива.

Помимо поиска минимального и максимального элементов в массиве, необходимо также хранить информацию об индексе строки, где находятся соответствующие элементы.

Когда находим элемент, меньший (больший) текущего экстремума, не только меняем значение экстремума, но и сохраняем номер строки, содержащий этот элемент:

Листинг 1.10.

```
1 ...
2 for (int i = 0; i < n; i++){
3   min = a[i][0];              //начальный min в i-ой строке
4   for (int j = 1; j < m; j++)
5     if ( a[i][j] > min){
6       min = a[i][j];        //определяем min
7       n_min = i;            //сохраняем номер строки с min
8     }
9 }
```

В случае, если минимальных или максимальных элементов массива больше одного, то в данной задаче необходимо запомнить номер строк, содержащих первый и последний экстремум. Если в качестве поиска экстремума использовать строгое неравенство: $a < min$, то будет найден именно первый минимальный элемент, так как элемент, равный min, не удовлетворяет этому неравенству.

Если же рассматривать нестрогое неравенство вида: $a \leqslant min$, то будет найден номер последнего элемента, так как элемент, равный min, удовлетворяет соответствующему неравенству.

Листинг 1.11. Реализация задания (пример 1.4)

```
#include<iostream>
using namespace std;

int **create (int n, int m){           //создание массива
  int **a = new int *[n];             //выделение памяти под массив
  for (int i = 0; i < n; i++)
    a[i] = new int [m];
  for (int i = 0; i < n; i++)         //заполняем массив
    for (int j = 0; j < m; j++){
      cout << "a[" << i << "][";
      cout << j << "]=";
      cin >> a[i][j];
    }
  return a;
}

void print (int **a, int n, int m){            //вывод массива на экран
  for (int i = 0; i < n; i++, cout << endl)
    for (int j = 0; j < m; j++)
      cout << a[i][j] << " ";
  cout << endl;
}

void change (int **a, int n, int m){          //меняем строки местами
  int min = a[0][0], max = a[0][0];           //нач. знач. для min, max
  int n_min = 0, n_max = 0;                   //номера строк
  for (int i = 0; i < n; i++)
    for ( int j = 0; j < m; j++){
```

```
29        if (a[i][j] < min){                    //первый min
30          min = a[i][j];
31          n_min = i;                          //номер строки с min
32        }
33        if (a[i][j] >= max){                   //последний max
34          max = a[i][j];
35          n_max = i;                          //номер строки с max
36        }
37      }
38       for (int j = 0; j < m; j++)              //обмен строк
39       swap( a[n_min][j], a[n_max][j]);
40  }
41
42  int main()
43  {
44
45    int n, m;                                  //размерность массива
46    cout << "n = "; cin >> n;
47    cout << "m = "; cin >> m;
48    int **a = create (n, m);                   //создание массива
49    print(a, n, m);                            //вывод на экран
50    change(a, n, m);                           //обмен строк
51    print(a, n, m);                            //вывод на экран
52    return 0;
53  }
```

Рассмотрим работу листига 1.11 на примере массива размерностью 3×3 (Полужирным шрифтом выделены искомые элементы):

Массив:	Результат:
$\begin{pmatrix} 3 & \mathbf{1} & 9 \\ 2 & 1 & 5 \\ 1 & \mathbf{9} & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 9 & 3 \\ 2 & 1 & 5 \\ 3 & 1 & 9 \end{pmatrix}$

Пошаговая реализация представлена в таблице:

i	j	$a[i][j]$	min	n_min	max	n_max
0	0	3	3	0	3	0
0	1	1	1	0	3	0
0	2	9	1	0	9	0
1	0	2	1	0	9	0
1	1	1	1	0	9	0
1	2	5	1	0	9	0
2	0	1	1	0	9	0
2	1	9	1	0	9	2
2	2	3	1	0	9	2

В результате нулевая и вторая строки массива меняются местами. □

Пример 1.5. Поменять местами столбец, содержащий только четный элементы, со столбцом, содержащим только нечетные элементы. Если таких столбцов несколько, поменять первые столбцы, удовлетворяющие условиям.

Для того, чтобы найти первый столбец, содержащий только нечетные элементы, понадобится две переменных логического типа: первая переменная, в листинге 1.12 обозначена как `flag_on`, отвечает за поиск нужного столбца (равна `true`, если такой столбец уже найден, и `false` в противоположном случае); вторая переменная, обозначенная в листинге 1.12 как `flag_o`, отвечает за поиск четных элементов в текущем столбце (равна `true`, если в столбце нет четных элементов, и `false` в противоположном случае).

Для столбца, содержащего только четные элементы, все аналогично. Соответствующие переменные называются `flag_en` и `flag_e`.

Рассмотрим алгоритм для поиска первого столбца, содержащего только нечетные элементы (строки 30–40).

Изначально переменной `flag_on` присваивается значение `false` (строка 25 листинга 1.12).

Для каждого столбца (строка 27) выполняются следующие действия:

1. Переменной `flag_o` присваивается значение `true` (предполагаем, что в столбце все элементы нечетные.)

2. Если `flag_on` равно `false` (еще не нашли нужного столбца), то ищем четный элемент в текущем столбце. Если такой элемент найден (строка 31), присваиваем переменной `flag_o` значение `false` и прекращаем цикл (строки 33–34).

3. Если после выполнения строк 31–35 листинга 1.12 значение переменной `flag_o` осталось `true`, значит в текущем столбце нет четных элементов. Соответственно, переменной `n_odd` присваиваем номер текущего столбца и меняем значение переменной `flag_on` на `true` (так как найден нужный столбец) (строки 36–39).

В строках 42–51 рассмотрен поиск столбца, содержащего только четные элементы по аналогичному алгоритму.

Замена столбцов происходит, если переменные `flag_on` и `flag_en` равны `true` (строки 55–59). Если одно из значений равно `false`, выводим сообщение о том, какого столбца не найдено (строки 60–68).

Листинг 1.12. Реализация задания (пример 1.5)

```
1  #include<iostream>
2  using namespace std;
3
4  int **create (int n, int m){          //создание массива
5    int **a = new int *[n];           //выделение памяти под массив
6    for (int i = 0; i < n; i++)
7      a[i] = new int [m];
8    for (int i = 0; i < n; i++)       //заполняем массив
9      for (int j = 0; j < m; j++){
10       cout << "a[" << i << "][";
11       cout << j << "]=";
12       cin >> a[i][j];
13     }
14   return a;
15 }
16
17 void print (int **a, int n, int m){          //вывод массива на экран
18   for (int i = 0; i < n; i++, cout << endl)
19     for (int j = 0; j < m; j++)
20       cout << a[i][j] << " ";
21   cout << endl;
22 }
23
24 bool change (int **a, int n, int m){          //меняем столбцы местами
25   bool flag_on = false, flag_en = false;    //true - если нашли соотв. столцбы
```

```
26  int n_odd = 0, n_even = 0;                  // odd - нечетный, even - четный
27  for (int j = 0; j < m; j++){                //для каждого столбца
28    bool flag_o = true, flag_e = true;     //предпол., столбцы удовл. условию
29    /*---------проверка для нечетного столбца-------------------*/
30    if (!flag_on){                       //если не нашли столбца с нечет. эл-ми
31      for (int i = 0; i < n; i++)
32        if (!(a[i][j] % 2)){             //встретили четный элемент
33          flag_o = false;              //меняем флаг
34          break;
35        }
36        if(flag_o) {                     //если true - только нечетные элементы
37          flag_on = true;              //нашли нужный столбец
38          n_odd = j;                   //запоминаем номер столцба
39        }
40    }
41    /*------------проверка для четного столбца ---------------*/
42    if (!flag_en){                       //все аналогично поиску нечетного столбца
43      for (int i = 0; i < n; i++)
44        if (a[i][j] % 2){
45          flag_e = false;
46          break;
47        }
48        if(flag_e){
49          flag_en = true;
50          n_even = j;
51        }
52    }
53  }
54  /*-----------------замена-----------------------------------------*/
55  if (flag_on && flag_en){                 //если нашли нужные столбцы
56    for(int i = 0; i < n; i++)
57      swap( a[i][n_even], a[i][n_odd]);   //меняем их местами
58    return true;
59  }
60  else{                                //иначе выводим какой столбец не нашли
61    if (!flag_on && !flag_en)
62      cout << "Нет столбцов, удовл. условиям\n";
63    else if (!flag_on)
64        cout << "Нет столбцов, сод. только нечетные элементы\n";
```

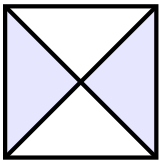
```
65      else
66        cout << "Нет столбцов, сод. только четные элементы\n";
67     return false;
68   }
69 }
70
71 int main()
72 {
73    setlocale (LC_ALL, "RUS");
74    int n, m;                                  //размерность массива
75    cout << "n = "; cin >> n;
76    cout << "m = "; cin >> m;
77    int **a = create (n, m);                   //создание массива
78    print (a, n, m);                           //вывод на экран
79    bool flag = change(a, n, m);               //обмен строк
80    if (flag)                                  //если обмен произошел
81      print (a, n, m);                        //вывод на экран
82    return 0;
83 }
```

Пример работы программы:

Массив:	Результат:
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 4 & 5 \\ 1 & 3 & 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 4 & 2 & 1 & 5 \\ 4 & 2 & 1 & 5 \\ 4 & 3 & 1 & 5 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 4 & 4 \\ 2 & 3 & 4 & 4 \end{pmatrix}$	Нет столбца, сод. только нечетные элементы
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 1 & 5 \\ 1 & 3 & 4 & 5 \end{pmatrix}$	Нет столбца, сод. только четные элементы
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$	Нет столбцов, удовл. условиям

□

Пример 1.6. Найти сумму четных элементов, расположенных в заштрихованной области, включая границы. Если четных элементов нет, вывести сообщение об этом.



Левый треугольник означает, что элемент массива расположен ниже главной диагонали и выше побочной. Согласно таблице 1.1 условие расположения элемента ниже главной диагонали $j \leqslant i$ (сама диагональ также включена в заштрихованную область). Условие расположения элемента выше побочной диагонали: $j \leqslant n - 1 - i$. Тогда условие нахождения элемента внутри заштрихованной области (левый треугольник):

$$j \leqslant i \vee j \leqslant n - 1 - i.$$

Правый треугольник означает, что элемент находится выше главной диагонали и ниже побочной диагонали. Соответственно, условие нахождения элемента внутри заштрихованной области (правый треугольник):

$$j \geqslant i \vee j \geqslant n - 1 - i.$$

Если n нечетное, то элемент `a[n/2][n/2]` принадлежит как правому, так и левому треугольнику (деление нацело, т. е., если `n = 5`, то `n/2 = 2`).

Например, если `n = 5`, то условие нахождения элемента в заштрихованной области:

$$\left[\begin{array}{l} j \leqslant i \wedge j \leqslant 4 - i, \\ j \geqslant i \wedge j \geqslant 4 - i. \end{array}\right.$$

Листинг 1.13. Реализация задания (пример 1.6)

```
#include<iostream>
using namespace std;

int **create (int n){            //создание массива
  int **a = new int *[n];          //выделение памяти под массив
  for (int i = 0; i < n; i++)
    a[i] = new int [n];
  for (int i = 0; i < n; i++)        //заполняем массив
```

```
9       for (int j = 0; j < n; j++){
10        cout << "a[" << i << "][";
11        cout << j << "]=";
12        cin >> a[i][j];
13      }
14    return a;
15  }
16
17  void print (int **a, int n){              //вывод массива на экран
18    for (int i = 0; i < n; i++, cout << endl)
19      for (int j = 0; j < n; j++)
20        cout << a[i][j] << " ";
21    cout << endl;
22  }
23
24  int summa (int **a, int n){                //сумма элементов
25     int k = 0, sum = 0;
26     for (int i = 0; i < n; i++)
27       for (int j = 0; j < n; j++)
28         if ((j <= i && j <= n - 1 - i)   //если элемент находится
29           || (j >= i && j >= n - 1 - i)){ // в заштрихованной области
30            if (!(a[i][j] % 2)){           //является четным
31              sum += a[i][j];             //ищем сумму
32              k++;                        //количество
33            }
34         }
35    if (!k) sum = -100000;                  // если четных элементов нет
36    return sum;
37  }
38
39  int main()
40  {
41      setlocale (LC_ALL, "RUS");
42    int n;                                  //размерность массива
43    cout << "n = "; cin >> n;
44    int **a = create (n);                   //создание массива
45    print(a, n);                            //вывод на экран
46    int sum = summa (a, n);
47    if (sum != -100000)
```

```
48      cout << "sum = " << sum << endl;
49    else
50      cout << "четных элементов нет\n";
51    return 0;
52  }
```

Пример работы программы:

Массив:	Результат:
$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & \mathbf{2} & 3 & \mathbf{4} & 5 \\ 1 & 3 & \mathbf{2} & \mathbf{4} & 5 \\ 1 & \mathbf{2} & 3 & \mathbf{4} & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$	18

□