**Двумерные массивы**

На сегодняшнем занятии мы будем изучать двумерные массивы — те самые таблицы, которые очень часто возникают в реальной жизни. Иногда их также называют «матрицами», как в алгебре.

**Создание и заполнение двумерных массивов**

Решим такую задачу: нарисовать на экране квадратный флаг Туапсинского района, где цвета обозначены разными цифрами. Точнее, задача формулируется так: по заданному числу N вывести на экран квадратную таблицу размером N на N, где главная диагональ (идущая из левого верхнего угла в правый нижний) заполнена единицами, в верхней правой половине таблицы стоят нули, а в левой нижней – двойки.

Если при работе с векторами мы сначала считывали размер, а затем создавали вектор нужного размера, то при работе с двумерными массивами мы воспользуемся другим способом. Нам заранее нужно будет создать массив максимально допустимого по условию задачи размера. В этой задаче — 100 на 100.

Чтобы создать двумерный массив размером 100 на 100 чисел нужно написать

int a[100][100];

Здесь a — это название массива, число в первых квадратных скобках задает количество строк, а во вторых — столбцов в этом массиве. Нумерация строк и столбцов начинается с нуля, a[0][0] — это верхний левый элемент таблицы. Обращение к конкретному элементу матрицы делается почти так же, как к элементу одномерного массива. Только в матрице индекса два: номер строки и номер столбца. Каждый из индексов указывается в отдельных квадратных скобках.

Как и на предыдущем занятии, программу удобно разбить на три логических блока: чтение, обработка и вывод данных. В этой задаче ввод очень простой, достаточно считать число N — размер таблицы. Обработка будет выглядеть следующим образом: мы сделаем цикл по строкам (счетчик i), а внутри него – цикл по отдельным элементам очередной строки (этот счетчик будет называться j). Для элементов, лежащих на диагонали, индексы i и j будут совпадать. В правой верхней части массива номер строки будет меньше номера столбца, а в левой нижней — наоборот, номер строки будет больше номера столбца.

for (int i = 0; i < n; i++) { //перебор строк

    for (int j = 0; j < n; j++) { //перебор столбцов

        if (i == j) {

            a[i][j] = 1;

        } else if (i < j) {

            a[i][j] = 0;

        } else {

            a[i][j] = 2;

        }

    }

}

//вывод

for (int i = 0; i < n; i++) { //перебор строк

    for (int j = 0; j < n; j++) { //вывод одной строки

        cout << a[i][j] << " ";

    }

    cout << endl; //перевод строки после того, как выведены все элементы

}

**Поле для сапера**

Игра «Сапер» известна многим. В некоторых клетках на прямоугольном поле лежат мины, а в остальных клетках — числа, обозначающие количество мин, которые окружают клетку (от 0 до 8). В нашей задаче по известному расположению мин на поле необходимо расставить числа во все свободные клетки, а на месте мин выводить «звездочку».

Как обычно, разделим задачу на стандартные части: чтение, обработка и вывод.

Пусть данные задаются в таком виде: в первой строке задаются два числа N и M — количество строк и столбцов соответственно. Оба этих числа не превосходят 100. После этого идет описание поля, состоящее из N строк. В каждой из строк содержится M чисел. Если очередное число равно 1, то в этой клетке стоит мина, а если 0 — мины нет.

В нашей программе мы считаем все поле в двумерный массив. Вот так будет выглядеть чтение:

int n, m;

cin >> n >> m;

int mines[100][100];

// чтение

for (int i = 0; i < n; i++) {

    for (int j = 0; j < m; j++) {

        cin >> mines[i][j];

    }

}

Следующий этап — обработка массива. В нашем случае основная обработкасостоит в том, чтобы для каждой свободной клетки посмотреть на 8 соседних и посчитать, в скольких из них есть мины. Это количество мы запомним в таком же по размеру массиве ответа.

Сразу подумаем о возможных проблемах. Смотря на соседей угловой или находящейся у края поля клетки, мы можем вылезти за пределы поля (то есть выйти за границы массива). Можно при просмотре каждого соседа писать условие с проверкой, лежит ли этот сосед в пределах поля. Но тогда нам придется писать 8 if’ов, в которых легко ошибиться. Поэтому можно поступить по-другому: окружить поле рамкой из свободных клеток. Они не повлияют на числа в клетках, а выхода за пределы массива не будет. Для этого нужно увеличить размеры массива по каждому из измерений на 2 (амка сверху и снизу, слева и справа от поля) и сразу заполнить его нулями. Входные данные тогда записываются, начиная с ячейки [1][1], а не [0][0].

Лучше произвести все эти действия на этапе чтения. После всех изменений чтение будет выглядеть так:

int n, m;

cin >> n >> m;

int mines[102][102]

for (int i = 0; i <= n + 1; i++) {

    for (int j = 0; j <= m + 1; j++) {

        mines[i][j] = 0;

    }

}

// чтение

for (int i = 1; i <= n; i++) {

    for (int j = 1; j <= m; j++) {

        cin >> mines[i][j];

    }

}

Перейдем к реализации обработки массива. Для каждой клетки поля нужно посмотреть на 8 соседей. Можно описать координаты каждого соседа отдельно, тогда у нас будет 8 почти одинаковых строк, а можно сделать очень классную вещь, чтобы писать меньше букв и допускать меньше ошибок. Пусть наша клетка имеет координаты [i][j]. Тогда ее соседями будут 8 клеток с координатами [i + 1][j - 1], [i + 1][j], [i + 1][j + 1], [i][j - 1], [i][j - 1], [i - 1][j - 1], [i - 1][j], [i - 1][j + 1]. Мы можем описать смещения относительно текущей клетки по каждой из координат в виде двух массивов dx и dy, каждый из которых состоит из 8 элементов, а возможные значения каждого из элементов это 0, 1 и -1. Тогда можно будет для каждой клетки поля пройти еще одним циклом по 8 соседям и просто посчитать сумму в них. Запишем это в виде программы:

int ans[102][102];

for (int i = 1; i <= n; i++) {

    for (int j = 1; j <= m; j++) {

        // координаты соседей (сдвиги)

        int dx[8] = {1, 1, 1, 0, 0, -1, -1, -1};

        int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};

        // перебор соседей

        int temp = 0;

        for (int k = 0; k < 8; k++)

            temp += mines[i + dy[k]][j + dx[k]];

        ans[i][j] = temp;

    }

}

Из новых языковых конструкций в этом коде у нас появилась инициализация массива конкретными значениями. Так заполняются массивы dx и dy.

В программе осталось написать только вывод. Он очень похож на предыдущую задачу, только нужно не забыть про вывод звездочки для тех клеток, где стоит мина.

// вывод

for (int i = 1; i <= n; i++) {

    for (int j = 1; j <= m; j++) {

        if (mines[i][j] == 1) {

            cout << "\*";

        } else {

            cout << ans[i][j];

        }

    }

    cout << endl;

}

Эта программа получилась очень длинной и достаточно трудной для понимания и отладки. Чтобы писать программу по кусочкам, каждый из которых остается коротким и понятным, нужны функции, которые мы изучим на следующем занятии.

***Выход из нескольких уровней вложенности циклов. Пояснения***:

Дострочный выход сразу из всех вложенных циклов это довольно распространённая ситуация, когда, например, нужно прервать все циклы при первом обнаружении некоторого события, вместо того чтобы дожидаться окончания всех циклов, что очевидно влияет на скорости выболнения кода, и эти знания пригодятся вам при решении задач данного раздела. Проблема начинаестя с того, что многие пологают что команда break выведет сразу из нескольких циклов, тогда как  в реальности, она выводит только из одного уровня вложенности!!! Как же быть? Давайте рассмотрим самый распространнённый случай с двумя уронями вленности:

for (.....) {

for (....) {

if (....) {

.... (1) ....

break;

}

}

.... (2) ....

}

Как уже было сказано,  команда break прерывает самый вложенный цикл, но внешний цикл продолжит выполнение начиная с инструкции (2), сразу после тела вложенного цикла.  Существует три распросранённых варианта решения этой проблемы:

1) Выходим из внутреннего цикла по команде return 0;. Как вы уже наверно догадались, в этом случае мы выходим не только из всех циклов, но вообще из функции main! Поэтому, перед тем как выйти по команде return 0;необходимо выполнить все необходимые действия, например, вывести слово "NO" на поток вывода.

*Примечание*: в следующем разделе вы будете проходить функции, так вот иногда функции создают именно для того, чтобы из них можно было выйти в любой момент по команде return 0;но при этом остаться в функции main Но вернёмся к нашим помидорам. Итак вот шаблон выхода из циклов по команде return:

for (.....) {

for (....) {

if (....) {

.... ....

// вычисляете всё что надо вычислить

// выводите всё что надо вывести

return 0;

}

}

}

2)  Второй вариант - выходим из каждого уровня вложенности в отдельности по команде break. Тут может выручить дополнительная переменная-флаг.

bool stop = false;

for (.....) {

for (....) {

if (....) {

.... (1) ....

stop = true;

break;

}

}

if (stop) break;

.... (2) ....

}

if (stop) {

.......

else {

......

}

3) Есть такое движение, выступающее против команды break. Они утверждают что во всех ситуациях где используется эта команда, можно обойтись и без неё.  Действительно, флаг прерывания цикла можно включить в условия циклов:

bool stop = false;

for (..;--- && !stop;..) { // --- некоторое логическое условие цикла

for (..;--- && !stop;..) { // --- некоторое логическое условие цикла

if (....) {

.... (1) ....

stop = true;

continue;

}

....

}

.... (2) ....

if (stop) { continue; }

.....

}

if (stop) {

.......

else {

......

}

Как видите в этом варианте все циклы закончатся один за другим, как только будет выставлен флаг stop во вложенном цикле. Инструкция continue тут используется дважды на случай, если в низу тела каждого из циклов есть ещё инструкции, выполнение которых нам нужно избежать, таким образом достигается почти немедленный выход из цикла. Но обратите внимание на то, что дополнительная проверка условия !stop будет происходить в условии цикла на каждом шаге обоих циклов.

4) Есть ещё один подход из разряда вредных советов, когда вы в теле цикла изменяете переменные от которых зависит условие цикла так чтобы его остановить! Давайте рассмотрим простой пример, когда у нас цикл проходят по индексу i:

for (int i=0; i<n; i++)

{

.........

if (условие)

{

........

i=n;

}

}

Для того чтобы остановить такой цикл достаточно написать  в теле цикла i=n; Таким путём можно избежать некоторого нагромождения проверок в условиях циклов, которое мы наблюдали в предидущем варианте решения. Ещё этот приём можно комбинировать с командой break, если в теле вложенного цикла написать i=n где i - индекс внешнего цикла, и сразу за этим выйти из вложенного цикла по break. Снова нужно заметить что это не панацея от всех проблем указанных в предидущих решениях, кроме того,  найдутся те что скажут что в отличае от цикла while, переменную цикла for вообще неправильно изменять, потому что по определению цикл for обычно применяют тогда, когда нам известно точное число итераций, кроме того, изменение переменной цикла for в нескольких местах затрудняет контроль над её текущим значением, а это может привести к ошибкам! Но на всякое правило есть исключение.

5) Ну и наконце самый вредный совет - команда goto! Согласитесь, для выхода из вложенного цикла подходит как нельзя лучше! Для других целей - большой вопрос?

for (.....) {

for (....) {

if (....) {

.... ....

goto AFTER\_CYCLE;

}

}

}

AFTER\_CYCLE:

.....

.....