Динамическое распределение многомерных массивов

Выделение памяти в свободном хранилище для многомерного массива предполагает использование операции *new* в несколько более сложной форме, чем для одномерного массива.

Пример**.** Выделение памяти под двумерные массивы.

Пусть требуется создать двумерный динамический массив целых чисел размерностью *n\*k:*

...

int n, k, i;

int\*\*mas;

…

cin>>n; //n- число строк массива

cin>>k; // k – число столбцов массива

mas = new int\*[n]; //выделение памяти под n указателей на строку

// выделение памяти для каждой строки по числу столбцов k

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

mas[i] = new int[k];

…

for (i=0;i<n;i++) // освобождение памяти

delete mas[i];

delete [] mas;

Сначала необходимо с помощью операции *new* выделить память под *n* указателей. Выделенная память будет представлять собой вектор, элементом которого является указатель. При этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области размером, равным второй границе массива.



Рассмотрим задачу , в которой решим задачу матричной алгебры по поиску суммы элементов главной диагонали матрицы , с заданными в процессе работы программы размерами.

Надо отметить, что матрицу очень легко отождествить с двумерным массивом. Поэтому именно задачи матричной алгебры широко используются в учебных целях, для получения навыков работы с массивами. Далее в задачах этого мы типа будем применять термины матричной алгебры к двумерному массиву.

При этом следует помнить, что нумерация строк и столбцов массива начинается с 0.

// Листинг 21

//Определение суммы элементов главной диагонали динамического массива

#include < iostream.h >

void main()

{

int \*\*array;

int n, m;

int i, j, sum;

//выделение памяти для динамического массива

cout<<"Enter number of rows and number of columns\n";

cin>> n>> m; //ввод размеров массива

**array = new int\*[n]; //выделение памяти для массива указателей**

**for (i = 0; i < n; i++) //выделение памяти для строк массива**

**{**

**array[i] = new int[m];**

cout<<"Now enter all numbers\n";

for (j = 0; j < m; j++) //ввод элементов массива

cin>> array[i][j];

}

//Вывод массива на экран

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

{

for (j = 0; j < m; j++)

{

cout<<array[i][j]<<'\t';

}

cout<<endl;

}

cout<<endl;

// Вычисление суммы элементов главной диагонали

sum = 0;

for (i = 0; (i < n) && (i < m); i++)

sum += array[i][i];

cout<<"Sum of diagonal:"<< sum <<"\n";

**// Освобождение памяти**

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

**delete[] array[i];**

**delete[] array;**

}

Результат

Enter number of rows and number of columns

4

4

Now enter all numbers

1

2

3

4

5

6

7

8

9

11

12

13

14

15

16

17

1 2 3 4

5 6 7 8

9 11 12 13

14 15 16 17

Sum of diagonal:36

В этом примере необходимо выделить память под двумерный массив.

Объявление указателя

int \*\*array;

таким образом, указывает на то, что array - указатель на массив указателей.

Далее объявляем переменные для хранения размеров массива, которые введем в процессе работы программы, индексов и суммы. После следуют оператор для ввода размеров массива. Далее приступим к выделению места для динамического двумерного массива. Обратите внимание-размер массива мы задали в ходе выполнения программы, а не так, как задавали ее для статического массива, при его объявлении. Прежде всего выделяем память для массива указателей , по количеству строк в массиве, а потом, в цикле, выделяем память для каждой строки массива. Адреса строк располагаются в соответствующих элементах массива указателей.

А теперь в цикле инициализируем массив. Для удобства и наглядности имеет смысл вывести на экран введенный массив построчно, в виде матрицы. Мы будем иметь возможность оценить правильность полученного результата.

Суммируем элементы главной диагонали, и выводим результат.

После работы с данными, освобождаем память – сначала, в цикле, каждой строки, а уже после этого, массива указателей.

Решим еще одну задачу

// Листинг 22

//Дана действительная матрица m х n. Найти последовательность, //состоящую из наибольших значений строк исходной матрицы

#include <iostream.h>

#include <stdlib.h>

#include <time.h>

#include <iomanip.h>

int main()

{

//выделение места для динамического массива

//заполнение массива

srand(time(NULL));

int m, n, i, j;

int \*\*matr;

int \*b;

//cout << "Enter number of rows and number of columns\n";

//cin >> m >> n;

n = rand() % 10 + 1; //индексы - случайные числа

m = rand() % 10 + 1;

**matr = new int\*[m]; //выделение памяти под массив указателей**

cout << "m: " << m << ", n: " << n << '\n';

cout << "matrix:\n";

for (i = 0; i < m; i++)

{

**matr[i] = new int[n]; //выделение памяти для строк массива**

//заполнение массива случайными числами и вывод массива на экран

for (j = 0; j < n; j++)

{

matr[i][j] = rand() % 100;

cout <<setw(5) << matr[i][j] ;

}

cout << endl;

}

cout << endl;

**b = new int[m]; //массив наибольших значений строк**

for (i = 0; i < m; i++) //определение наибольших значений строк

{

b[i] = matr[i][0];

for (j = 1; j < n; j++)

{

if (matr[i][j] > b[i])

{

b[i] = matr[i][j];

}

}

}

cout << "Maximums of rows:\n";

for (i = 0; i < m; i++) {

cout << b[i] << endl;

}

**// Освобождение памяти**

**for (i = 0; i < m; i++)**

**delete[] matr[i];**

**delete[] matr;**

**delete[] b;**

return 0;

}

Результат

m: 5, n: 3

matrix:

69 50 48

97 72 11

49 56 95

5 92 55

30 72 88

Maximums of rows:

69

97

95

92

88

В этом примере мы снова применили наши знания программирования для решения задачи матричной алгебры.

В отличие от предыдущих задач, мы не вводили значения элементов массива вручную , а воспользовались генератором случайных чисел. Это очень удобно, особенно если предположить, что массив может быть велик, и ввод с клавиатуры сильно утомителен.

Чтобы работать с генератором случайных чисел, нужно подключить соответствующую библиотеку

#include <stdlib.h>

а в самой программе воспользуемся функцией *rand()* для генерации случайного числа.

Но, чтобы функция заработала, нужно выполнить инициализацию генератора случайных чисел. Инициализацию генератора случайных чисел лучше выполнить таймером, тогда значения будут обновляться.

Для работы с таймером также необходима библиотека

*#include <time.h>*

и соответствующий оператор для инициализации генератора случайных чисел

*srand(time(NULL));*

Случайным образом мы задаем индексы элементов массива

m = rand() % 10 + 1;

Производимое генератором число случайно, и находится в определенном , достаточно большом диапазоне. Для заполнения массива мы можем захотеть, и обычно хотим, этот диапазон ограничить.

Когда мы получаем остаток от деления любого числа на десять, например, результат всегда будет находиться в диапазоне от 0 до 9. Но поскольку мы задаем индекс массива логично задать его в интервале от 1 до 10, что мы и сделали, добавив единицу. Смещение интервала может быть любым, каким захотим.

При заполнении массива случайными числами, мы воспользовались оператором

*matr[i][j] = rand() % 100;*

Значения элементов динамического массива будут меньше ста.

В программе также будет создан еще один одномерный динамический массив

*b = new int[m];*

для хранения наибольших значений строк.

Он также создается динамически, так как его размер неизвестен до выполнения программы.

Остальные действия , предпринятые нами, для создания динамического двумерного массива, уже знакомы. Перед завершением программы главное не забыть об освобождении всей выделенной памяти.