8. Матрицы (двумерные массивы). Организация ввода и вывода через файлы

Из курса математики известно, что матрица представляет собой объект, который можно изобразить в виде прямоугольной таблицы, со строками и колонками. При этом каждый элемент матрицы имеет два индекса: первый означает номер его строки, второй — номер колонки (столбца). На языке C/C++ матрица (двумерный массив) представляется как «массив массивов», т. е. массив, компонентами которого являются другие массивы. Соответственно, вся работа с матрицами производится по правилам работы с одномерными массивами. В частности, следующее описание:

int a[3][4];

задаёт целочисленную матрицу размера 3×4 (с 3 строками и 4 столбцами). Элемент матрицы *aij* записывается при этом как a[i][j], оба индекса начинаются от нуля.

В памяти компьютера элементы матрицы располагаются связно и последовательно — развёрнуто по строкам. Учитывая это обстоятельство, объявление матрицы с инициализацией может происходить двояко: либо каждый из двух уровней массива заключается в свои фигурные скобки (в этом случае размеры при описании можно не указывать), либо задаётся общий список элементов в том порядке, в котором они располагаются в памяти. Так, эквивалентны следующие три описания:

int b[2][3]={{4,5,1},{2,6,3}};

int b[][]={{4,5,1},{2,6,3}};

int b[2][3]={4,5,1,2,6,3};

Вернёмся к описанной выше матрице *a*(3×4). Если элемент матрицы записывается как a[i][j], то a[i] (с одним индексом) — это наименование массива, представляющего i‑ю строку матрицы a, т. е. адрес её начала в памяти компьютера (указатель на её начальный элемент a[i][0]). Таким образом, выстраивается одномерный массив: a[0], a[1], a[2] из указателей на начало строк матрицы. Соответственно, его наименование a (без индексов вообще) — это есть адрес начала этого массива указателей, т. е. **указатель на указатель** (адрес указателя).

a[0][0]

a[0][1]

a[0][2]

a[0][3]

a[1][0]

a[1][1]

a[1][2]

a[1][3]

a[2][0]

a[2][1]

a[2][2]

a[2][3]

a[0]

a[1]

a[2]

a

Из этого, в частности, вытекает, что с помощью сложения указателя с целым числом и операции разыменования указателя элемент матрицы a[i][j] может ещё быть записан как \*(a[i]+j) либо \*(\*(a+i)+j).

**Выделение динамической памяти** под целочисленную (к примеру) матрицу *a* размера *m*×*n* производится в два этапа. Сначала выделяется *m* ячеек памяти под массив указателей на строки (*a*[0], *a*[1], … , *a*[*m*–1]), затем в цикле для каждого *a*[*i*] выделяется по *n* ячеек непосредственно для строк матрицы:

int \*\*a=new int\*[m];

for(int i=0;i<m;i++) a[i]=new int[n];

**Освобождение динамической памяти** производится в обратном порядке: сначала освобождается память, выделенная для строк, затем память, выделенная для указателей на строки:

for(int i=0;i<m;i++) delete[] a[i];

delete[] a;

Подобно тому, как работа с одномерными массивами предполагала активное использование циклов, работа с матрицами предполагает использование двойных циклов (конструкций «цикл в цикле»), с проходом по обоим измерениям. Продемонстрируем это на примере действий по вводу и выводу вещественной матрицы *a*(*m*×*n*).

Для ввода матрицы имеется несколько способов организации запросов (на всю матрицу, на каждую строку, на каждый элемент). Нам представляется наиболее удобным ввод с запросом на каждую строку:

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

cout<<”введите ”<<i<<” строку из ”<<n

<<” чисел\n”;

for(int j=0;j<n;j++) cin>>a[i][j]; }

Для вывода матрицы, чтобы она вышла красиво, в виде таблицы с выровненными колонками, предпочтительно использовать форматированный вывод в стиле C. При этом каждая строка матрицы печатается на одной экранной строке, по окончанию строки, перед печатью следующей, на экране делается переход на новую строчку:

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

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

printf(”%7.2f”,a[i][j]);

printf(’\n’); }

Число позиций в спецификаторе %7.2f здесь стоит условное. На практике следует учитывать предполагаемую величину элементов матрицы и требуемую точность, а также то, что между соседними элементами должен быть хотя бы один пробел.

***Внимание!*** При организации двойных циклов прохода матрицы следует иметь в виду следующее. Первый индекс — это номер строки, но проход строки — это проход второго индекса, при первом фиксированном. Аналогично, второй индекс — это номер столбца, а проход столбца — это проход первого индекса при втором фиксированном.

**Пример 1.** В каждом столбце целочисленной матрицы *a*(4×5) переставить первый элемент с последним, если первый элемент больше последнего.

#include <stdio.h>

int main() {

const int M=4, N=5;

int a[M][N];

int i, j, c;

// ввод матрицы

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

printf(”задайте %d строку\n”,i);

for(j=0;j<N;j++) scanf(”%d”,&a[i][j]);}

// преобразование матрицы

for(j=0;j<N;j++) if(a[0][j]>a[N-1][j]

{c=a[0][j];

a[0][j]= a[N-1][j];

a[N-1][j]=c;}

// вывод результатов

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

for(j=0;j<N;j++) printf(”%4d”,a[i][j]);

printf(’\n’); }

return 0; }

**Пример 2.** Даны: *n*, квадратная матица *a*(*n*×*n*) и массив *b*(*n*). Заменить в матрице *a* все строки с нечётными номерами на массив *b*.

Поскольку тип элементов в задаче не оговорен и не вытекает из контекста, для написания программы его можно брать любым из числовых. Положим, что и матрица, и массив относятся к типу double.

#include <stdio.h>

int main() {

int n,i,j;

// ввод n

printf(”задайте n\n”);

scanf(”%d”,&n);

// выделение памяти

double \*\*a=new double\*[n];

for(i=0;i<n;i++) a[i]=new double[n];

double \*b=new double[n];

// ввод матрицы и массива

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

printf(”задайте %d строку матрицы\n”,i);

for(j=0;j<n;j++) scanf(”%lf”,&a[i][j]);}

printf(”задайте массив\n”);

for(i=0;i<n;i++) scanf(”%lf”,&b[i];)

// преобразование матрицы

for(i=1;i<n;i+=2)

for(j=0;j<n;j++) a[i][j]=b[j];

// вывод результатов

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

for(j=0;j<n;j++) printf(”%9.3lf”,a[i][j]);

printf(’\n’); }

// освобождение памяти

for(i=0;i<n;i++) delete[] a[i];

delete[] a;

delete[] b;

return 0; }

Если матрица имеет большие размеры, то ввод данных с клавиатуры может быть достаточно трудоёмким. Особенно если проводится серия пробных запусков программы с целью её отладки. В этом случае удобно, когда чтение исходных данных производится программой в автоматическом режиме, из заранее заготовленных файлов. С другой стороны, результаты работы программы в ряде случаев тоже предпочтительно сохранять в отдельных файлах, для возможного дальнейшего использования. В C++ имеются две возможности чтения и записи с использованием файлов: в стиле C и потоковый ввод/вывод. Мы рассмотрим оба варианта.

**Ввод/вывод через файлы в стиле C** предполагает подключение заголовочного файла stdio.h. При вводе/выводе данные рассматриваются как поток байтов, который связывается с каким-либо устройством или файлом. Для работы с потоком он должен быть открыт в каком-либо режиме (для чтения или записи). Поток открывается при участии **файлового указателя** на структуру стандартного типа FILE, содержащую всю необходимую для работы с потоком информацию. Действие по открытию потока записывается так:

*файл\_указ*=fopen(”*имя\_файла*”,”*реж\_откр*”);

Здесь *файл\_указ* — это файловый указатель (типа FILE\*), который используется при дальнейших операциях с потоком, *имя\_файла* — имя файла, с которым связывается поток (при размещении его в другом месте относительно текста программы требуется указывать к нему путь), *реж\_откр* — режим открытия, для которого имеется несколько вариантов, назовём самые популярные:

* ”r” — файл открывается для чтения;
* ”w” — открывается пустой файл для записи (если файл существует, то он стирается);
* ”a” — файл открывается для добавления записей в его конец.

После открытия файла можно либо из него читать данные, либо записывать в него информацию. Операции чтения из файла и записи в файл по своему виду похожи на соответствующие операции чтения с клавиатуры и печати на монитор. Операция чтения из файла записывается так:

fscanf(*файл\_указ*,”*строка\_форм.”*,*список\_адресов*);

Операция записи в файл:

fprintf(*файл\_указ*,”*стр.\_с\_форм.”*,*список\_вывода*);

***Внимание!*** Перед запуском программы, с чтением данных из файла следует подготовить файл данных, записав и сохранив в нём блок исходных данных, расположенных в порядке чтения в программе. Зато в самой программе при чтении исходных данных отсутствует необходимость в высвечивании запросов-подсказок, поскольку ввод данных будет производиться автоматически.

Для **ввода/вывода с помощью потоков C++** подключаем заголовочный файл fstream, с подключением к пространству имён std. После чего определяем имена входного и выходного потоков и связываем их с соответствующими файлами. Удобно, по аналогии со стандартными потоками cin и cout, наименовать потоки для работы с файлами как, соответственно, fin и fout. В этом случае связывание потоков с файлами выглядит следующим образом:

ifstream fin(”*имя\_файла\_данных*”);

ofstream fout(”*имя\_файла\_результатов*”);

Действие по вводу и выводу с помощью файловых потоков записывается так же, как и для потоков cin и cout, через операции >> и <<:

fin>>*величина*>>*величина*>>…;

fout<<*элемент\_вывода*<<*элемент\_вывода*<<…;

Однако в случае применения операции fout для матриц, её колонки могут оказаться невыровненными.

**Пример 3.** Написать программу умножения матриц *a*(*m*×*n*) и *b*(*n*×*k*) и выполнить её для



Исходные данные прочитать из файла. Результат записать в другой файл.

Пусть *c* – результирующая матрица. Тогда *c* имеет размеры *m*×*k*, и ее элементы вычисляются по формуле



Для реализации этой формулы нам потребуется тройной цикл: по *i* и *j* (проход по матрице *c*) и по *l* для подсчёта суммы при вычислении каждого *cij*.

Файл данных назовём data.txt, файл результатов — result.txt.

#include <stdio.h>

int main() {

int i,j,k,l,m,n;

// открытие файлов

FILE \*fi=fopen(”data.txt”,”r”);

FILE \*fo=fopen(”result.txt”,”w”);

// чтение размеров матриц

fscanf(fi,”%d%d%d*”*,&m,&n,&k);

// выделение памяти для матриц

float \*\*a=new float \*[m];

for(i=0;i<m;i++) a[i]=new float [n];

float \*\*b=new float \*[n];

for(i=0;i<n;i++) b[i]=new float [k];

float \*\*c=new float \*[m];

for(i=0;i<m;i++) c[i]=new float [k];

// чтение матриц a и b из файла

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

for(j=0;j<n;j++) fscanf(fi,”%f”,&a[i][j]);

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

for(j=0;j<k;j++) fscanf(fi,”%f”,&b[i][j]);

// вычисление матрицы c

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

for(j=0;j<k;j++) {

c[i][j]=0;

for(l=0;l<n;l++)c[i][j]+=a[i][l]\*b[l][j];

}

// печать матрицы c в файл

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

for(j=0;j<k;j++)fprintf(fo,”%5.1f”,c[i][j]);

fprintf(fo,”\n”); }

// освобождение памяти

for(i=0;i<n;i++) delete[] a[i];

delete[] a;

for(i=0;i<k;i++) delete[] b[i];

delete[] b;

for(i=0;i<k;i++) delete[] c[i];

delete[] c;

return 0;}

В файл данных result.txt (который должен быть создан перед первым запуском программы) надо занести исходные данные в порядке чтения в программе, а именно: сначала размеры *m*, *n*, *k*, затем матрицу *a*, затем матрицу *b*:

3 3 2

4 0 –1

7 –2 –4

5 7 9

-3 4

-4 –7

1 7

**Пример 4.** Дана целочисленная матрица *a*(5×10). Сформировать массив *b* из сумм элементов столбцов матрицы *a*. Исходные данные прочитать из файла. Результирующий массив записать в тот же файл, в виде строки, отделённой от исходной матрицы пустой строчкой.

#include <stdio.h>

int main() {

const int m=5, n=10;

int a[m][n];

int b[n]={}; // массив изначально нулевой

int i,j;

FILE \*f=fopen(”data.txt”,”r”); // откр. файла

// на чтение

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

for(j=0;j<n;j++) fscanf(f,”%d”,&a[i][j]);

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

for(i=0;i<m;i++) b[j]+=a[i][j];

f=fopen(”data.txt ”,”a”); // повторное откр.

// файла на дозапись

fprintf(f,”\n”);

for(i=0;i<n;i++) fprintf(f,”%3d”,b[i]);

return 0; }

**Задачи**

1. Ввести матрицу *a*(5×4). Найти:

а) сумму всех элементов;

б) сумму элементов строки с номером 3;

в) наибольший элемент столбца с номером 2.

1. Дана матрица *a*(5×5). Заменить в ней нулями элементы под главной диагональю (весь нижний треугольник).
2. Преобразовать матрицу *a*(*n*×*n*) к транспонированному виду (отразить от главной диагонали). Вторую матрицу не использовать.
3. Отразить матрицу *a*(*n*×*n*) относительно побочной диагонали. Вторую матрицу не использовать.
4. Даны: *M*, *N*, матрица размера . Найти максимальный среди минимальных элементов ее строк.
5. Даны: *M*, *N*, целочисленная матрица размера . Найти номер первой из ее строк, содержащих равное количество положительных и отрицательных элементов (нулевые элементы матрицы не учитываются). Если таких строк нет, то вывести –1.
6. Даны: *m*, *n*, матрица *a*(*m*×*n*) и число *x*. Заменить единицами все элементы матрицы *a*, превосходящие *x*. Исходные данные прочитать из файла. Результаты записать в другой файл.
7. Даны: *m*, *n*, матрица *a*(*m*×*n*). Сформировать массив *b* из наибольших элементов её строк. Исходные данные прочитать из файла. Результаты записать в другой файл, в виде столбца.
8. Даны: *N*, матрица . Если хотя бы один элемент строки матрицы отрицателен, то все элементы этой строки заменить нулями. Исходные данные прочитать из файла. Результаты записать в тот же файл, отделив от исходной матрицы пустой строкой.
9. Даны: *M*, *N*, матрица размера . Найти количество её строк, элементы которых упорядочены по возрастанию. Исходные данные прочитать из файла. Результаты записать в тот же файл, отделив от исходной матрицы пустой строкой.