Нижегородский государственный университет

Производные типы данных: массивы

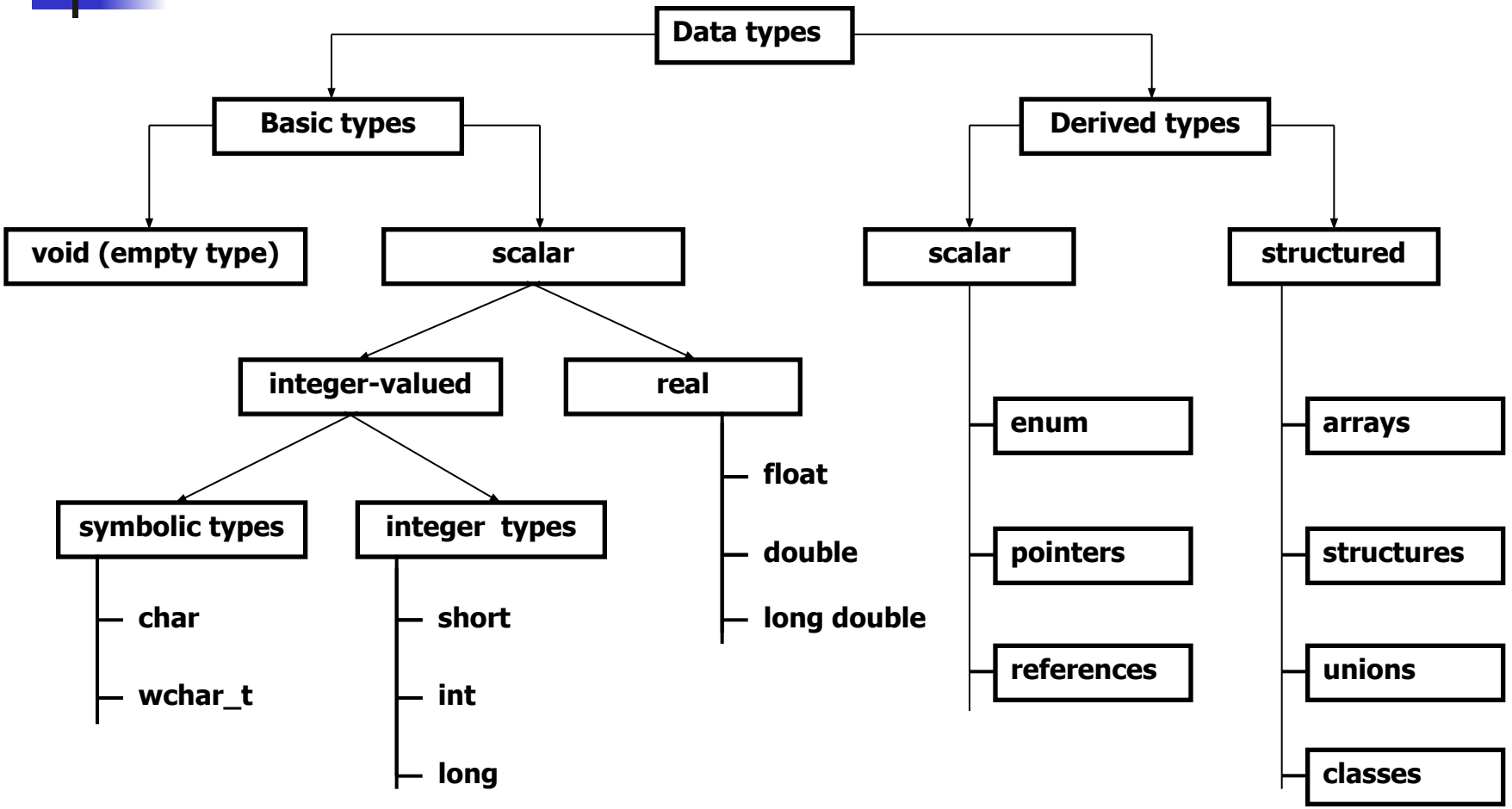
В.А.Гришагин

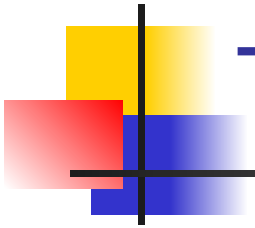
Институт информационны технологий, математики и механики

Нижний Новгород, Россия
2020

Models a
Prog

Типы данных


Data types
Basic types
Derived types
void (empty type)
scalar
scalar
structured
integer-valued
real
enum
arrays
symbolic types
integer types
float
pointers
structures
char
short
double
references
unions
wchar_t
int
long double
long
classes



Тип данных ***массив*** (***array***)

Использование скалярных типов неудобно для представления и обработки данных значительных объемов. Часто мы имеем дело с данными, объединенными вместе в одну группу: списки, таблицы, регистры и т.п., причем в группе отдельные элементы данных являются скалярными и имеют идентичные характеристики (за исключением значений), например, все являются целыми числами. Если в группе много элементов, то заводить для каждого отдельную переменную нереально. В этой ситуации язык предлагает специальный тип данных, называемый ***массивом*** (array).

Массив – это множество скалярных данных одного типа, имеющих общее имя (идентификатор массива) и отличающихся друг от друга согласно некоторой системе нумерации элементов массива, в соответствии с которой реализуется доступ к элементам. Номер элемента в массиве называется его ***индексом***.

Элементы массива занимают непрерывную область памяти и располагаются последовательно один вслед за другим согласно индексам.

Операция [ ]

Если компоненты массива нумеруются одним индексом, такой массив называется одномерным, или вектором, если два индекса – двумерным, или матрицей, N>2 индексов – N-мерным.

Как объявить массив? Для этой цели предназначена операция **[ ]**.

В случае одномерного массива общая форма объявления имеет вид

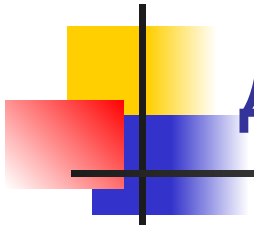
dattp namearr[numel];

где ***dattp*** - тип данных элементов массива,

namearr - идентификатор массива,

numel - количество элементов массива.

Таким образом, использование квадратных скобок в декларативном операторе после идентификатора означает, что этот идентификатор рассматривается как массив типа ***dattp*** и является ОБЩИМ именем всех элементов массива.



Доступ к элементам массива

Все элементы массива нумеруются целыми числами от **0** до **numel-1**, где ***numel*** - это число элементов, указанных при объявлении массива. Обратите внимание: первый элемент массива имеет номер **0**!

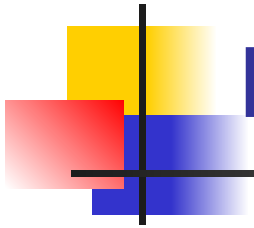
Для получения доступа к конкретному элементу массива (чтобы выполнять с ним вычислительные операции или операции ввода/вывода) используется НОМЕР этого элемента в массиве, заключенный в квадратные скобки [ ].

Примечание. Операция [ ] имеет различное назначение в декларативном и исполнительном операторах. В декларативных операторах она ОПРЕДЕЛЯЕТ тип данных массив. В исполнительном операторе она обеспечивает ДОСТУП к элементу массива.

Пример

```
int mas[10], a[2], f;
long double s, ldm[25];
.   .   .
    mas[0]=23+a[1];
    s=ldm[12];
    f=mas[10];
```

Компилятор не отслеживает ситуацию, когда индекс превосходит максимальное значение **numel-1** . Вы даже не получаете предупреждения об этой ситуации. За этим приходится следить самому программисту.



Инициализация массива

Инициализация массива означает назначение конкретных значений его элементами при объявлении. Массивы могут быть инициализированы списком констант или константных выражений того же самого типа, что и тип массива. Элементы в списке инициализации разделяются запятыми, а список должен быть заключен в фигурные скобки.

Пример.

Инициализация массива количеством дней в каждом из месяцев не високосного года.

```
int days[12]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Инициализация массива

Если список инициализации короче количества элементов массива, первые компоненты массива инициализируются элементами списка, а оставшиеся нулями, если массив числовой, или пустыми константами, если массив символьный.

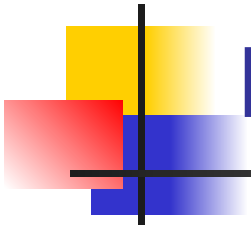
Массив может быть инициализирован без явного указания количества элементов в нем. В этом случае количество элементов будет равно количеству констант в списке инициализации.

char code[]={‘a’, ‘b’, ‘c’};

Если массив явно не инициализируется, то массивы класса памяти ***static*** и ***external*** инициализируются нулями, а элементы массивов классов памяти ***auto*** и ***register*** не инициализируются, и их значения не определены.

Как найти количество ***numel*** элементов в массиве **mas** типа **mt**, у которого при объявлении оно явно не указано?

numel = sizeof(mas)/sizeof (mt)



Массивы и указатели

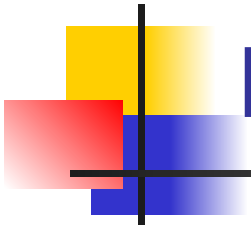
Имя массива (без квадратных скобок!) является УКАЗАТЕЛЕМ типа ***pointer*** , который указывает начальный адрес участка памяти, выделенного массиву, т.е. адрес младшего байта памяти, занимаемой первым элементом массива.

Так как имя массива является указателем, мы можем применять к нему любые операции работы с указателями.

К примеру, объявим массив ***mas***:

int mas[4];

Здесь идентификатор ***mas*** это указатель (на тип ***int***). Что такое****mas*** ? Это значение элемента с адресом mas, т.е. ***mas[0]*** ! К элементу ***mas[0]*** мы можем применить операцию **&**: ***&mas[0]***. Результат будет адресом этого элемента в памяти, а именно, ***mas*** !



Массивы и указатели

Мы можем добавить к указателю ***mas*** целое число:

mas+2

Что получится? Согласно правилам операций с указателями, мы не добавляем к адресу ***mas*** число 2, мы добавляем к нему ***2 единицы памяти*** отипа int. Тогда результат – это адрес

mas + 2*sizeof(int) = &mas[0]+2*sizeof(int)

Но это ***&mas[2]*** – адрес элемента ***mas[2]*** ! Как результат, чтобы получить доступ к элементу с индексом ***n*** , можно использовать любую из следующих конструкций:

mas[n] или ****(mas+n)***

Массивы и указатели

```
int mas[4]= {8, 2, 4};
int *pt;
pt=mas+1;
```

mas = 0100

mas[0] = 8

&mas[0] = 0100

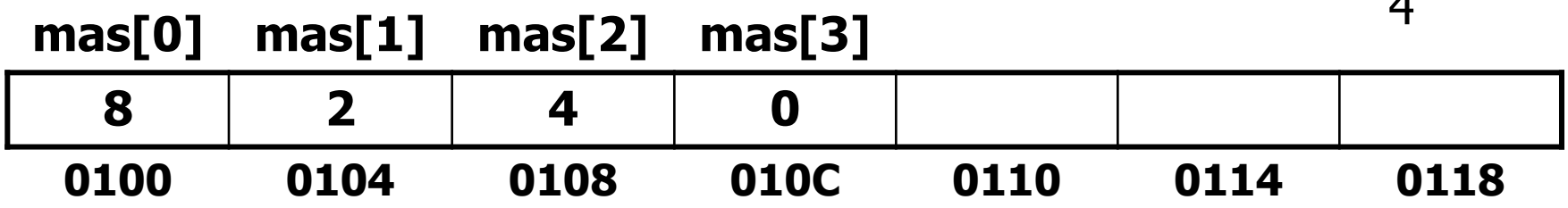
&mas[2] = 0108

mas+2 = 0108

*(mas+2) = 4

*mas+2 = 10

*(pt+1) = 4



Массивы и указатели

```
#include "stdafx.h"
#include <conio.h>
using namespace std;

int main()
{
      int mas[4]={8,2,4,15};
      int i,  *pt;
      cout<<"\naddrmas="<<mas<<"\n";
      pt=mas;
      for (i=0;i<=3;i++)
      {
        cout<<"\naddress of the "<<i<<"-th element="<<&mas[i];
            cout<<"\naddress of the "<<i<<"-th element="<<mas+i;
        cout<<"\naddress of the "<<i<<"-th element="<<pt<<"\n";
        pt=pt+1;
      }
```

Массивы и указатели

```
    pt=mas;
    for (i=0;i<=3;i++)
    {
       cout<<"\nvalue of the "<<i<<"-th element="<<mas[i];
       cout<<"\nvalue of the "<<i<<"-th element="<<*(mas+i);
       cout<<"\nvalue of the "<<i<<"-th element="<<*pt<<"\n";
       pt=pt+1;
    }
    getch();
}
```

Массивы и указатели

```
addrmas=0012F034

address of the 0 element=0012F034
address of the 0 element=0012F034
address of the 0 element=0012F034

address of the 1 element=0012F038
address of the 1 element=0012F038
address of the 1 element=0012F038

address of the 2 element=0012F03C
address of the 2 element=0012F03C
address of the 2 element=0012F03C

address of the 3 element=0012F040
address of the 3 element=0012F040
address of the 3 element=0012F040
```

Массивы и указатели

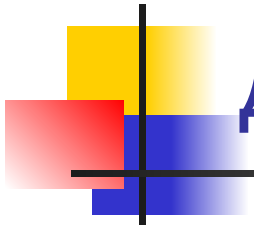
```
value of the 0 element=8
value of the 0 element=8
value of the 0 element=8

value of the 1 element=2
value of the 1 element=2
value of the 1 element=2

value of the 2 element=4
value of the 2 element=4
value of the 2 element=4

value of the 3 element=15
value of the 3 element=15
value of the 3 element=15
```

Многомерные массивы

На практике данные часто организованы в виде прямоугольных таблиц, или матриц, в которых элементы имеют одинаковый тип.

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$$

Как описывать матрицы на языке C? Очевидно, что каждая строка может быть описана как массив. Однако все строки организованы одинаково: они содержат одно и то же число элементов одинакового типа. Это означает, что множество строк также можно рассматривать как массив!



Двумерные массивы

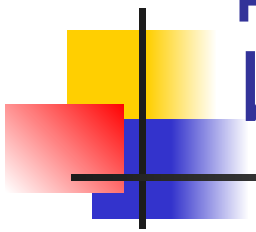
Следовательно, правила языка позволяют записать конструкцию типа

dattp mas[n] [k] ;

чтобы объявить двумерный массив **mas** с **n** строками и **k** столбцами, содержащими компоненты типа **dattp** (**n** и **k** должны быть целыми константами).

Другими словами, мы объявили массив **mas** , состоящий из **n** элементов, каждый из которых является МАССИВОМ, состоящим из **k** элементов (уже скалярных).

Двумерные массивы. Инициализация.



Как инициализировать двумерный массив? Для одномерных массивов мы заключали список констант в фигурные скобки. То же самое правило действует и для двумерных массивов, учитывая, что двумерный массив является одномерным массивом одномерных строк-массивов:

```
dattp mas[n] [k] = { {список компонент 1-й строки},
                     {список компонент 2-й строки},
                              .   .   .
                      {список компонент n-й строки}
                   };
```

Двумерные массивы. Инициализация.

Example.

```
float  fmas[3][3] = { { 2.0,3.0,-1.0 },
                      { 7.1,4.51, 1.2 },
                      { -2.2, 3.4567,2.5}
                    };
```

Правила, относящиеся к ситуации, когда не хватает констант инициализации, аналогичны случаю одномерных массивов: недостающие компоненты заполняются нулями. Если же число констант в списке превосходит длину строки (или количество строк больше указанного при объявлении), компилятор выдает сообщение об ошибке.

Если массив не инициализируется при объявлении, то при использовании спецификатора static он заполняется нулями, а в случае спецификаторов auto или register значения его элементов не определены.

Двумерные массивы. Распределение памяти.

Вспомним, как располагаются элементы одномерного массива в памяти. Они занимают непрерывный сегмент ячеек и размещаются один вслед за другим. При этом каждый элемент занимает одну или несколько ячеек (байтов) в зависимости от типа массива.

Давайте теперь возьмем двумерный массив

dattp mas[n] [k] ;

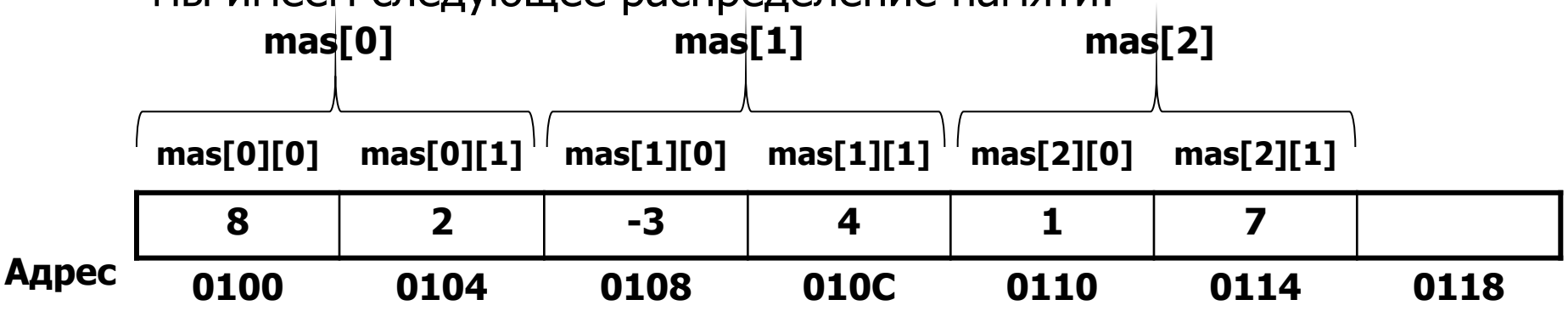
Он представляет собой одномерный массив одномерных массивов-строк, состоящих из ***k*** элементов типа ***dattp***. Это означает, что сначала располагается первый элемент-строка ***mas[0]***, затем элемент-строка ***mas[1]*** и т.д. вплоть до элемента строки ***mas[n-1]***.

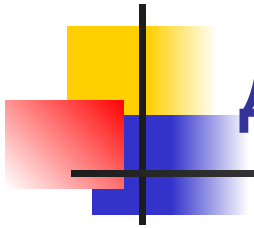
Двумерные массивы. Распределение памяти.

Но каждый элемент ***mas[i]*** является одномерным массивом и состоит из ***k*** элементов ***mas[i][0], mas[i][1], . . . , mas[i][k-1],*** которые также располагаются последовательно. Таким образом, для конкретного массива

```
int mas[3] [2] ={{8, 2},
                 {-3, 4},
                 {1, 7}
                };
```

мы имеем следующее распределение памяти:





Динамические массивы

До сих пор мы рассматривали массивы, чьи размеры (количество компонент в размерностях) были строго определены при объявлении массива и количество компонент могло быть только константой. Во многих случаях это неудобно, потому что если вы хотите иметь дело с массивом другого размера, вы должны переписать оператор объявления и выполнить компиляцию программы заново.

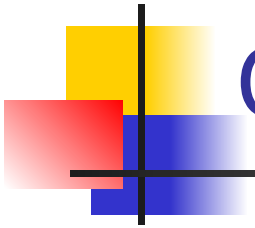
Чтобы преодолеть эти ограничения, в языке имеется возможность работать с массивами, размеры которых не определены заранее, а задаются в процессе работы программы. Такие массивы называются ДИНАМИЧЕСКИМИ.

Операции с динамической памятью

Реализация таких массивов базируется на специальном механизме работы с памятью, называемой ДИНАМИЧЕСКОЙ ПАМЯТЬЮ. Этот механизм позволяет программе запросить из динамического пула нужный объем дополнительной памяти по мере необходимости для создания некоторого информационного объекта, называемого динамическим, и затем, если эта память больше не нужна, вернуть ее обратно.

Для взаимодействия с динамической памятью, порождения и ликвидации динамических объектов используются разные способы, рассмотрим один из них, основанный на ***new*** и ***delete***.

Операция ***new*** запрашивает и получает память для создаваемого динамического объекта. Результатом является УКАЗАТЕЛЬ на начало выделенного участка памяти. Таким образом, для создания динамического объекта необходимо создать связанный с ним указатель и присвоить ему значение операции ***new*** .



Операции с динамической памятью

Операция ***new*** в качестве своего аргумента требует указать объем запрашиваемой для объекта памяти. Это достигается просто указанием ТИПА создаваемого объекта.

Пример. Создаем динамическую переменную типа ***int***.

```
int *p;  //определяем указатель на тип int
p=new int;  // получаем память (4 байта), доступ к ней через указатель p
```

Более компактно:

```
int *p=new int;  //Объявление указателя совмещено с запросом памяти (фактически функция new выполняет инициализацию указателя)
```

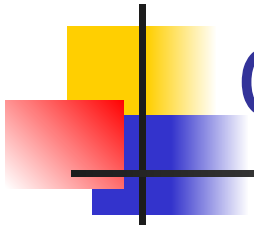
Операции с динамической памятью

Если мы запрашиваем память для одномерного массива, то его тип имеет вид ***mtp[n]***, где ***mtp*** - тип элементов массива, а ***n*** – количество компонент. Тогда аргументом функции ***new*** будет тип ***mtp[n]*** ! При этом количество элементов ***n*** может задаваться целочисленной ПЕРЕМЕННОЙ с положительным значением и даже целочисленным положительным выражением!

Если динамический объект больше не нужен, надо вернуть занятую им память в общий динамический пул с помощью операции ***delete***, аргументом которой является указатель на динамический объект, если объект не массив, а в случае массива – оператор

```
delete [] mas;
```

где ***mas*** – имя массива (или, что одно и то же, указатель на массив).



Операции с динамической памятью

Пример 1

```
int n, *mas;
.  .  .
cin>>n;
mas=new int[n];
.  .  .
delete [ ] mas;
.  .  .
```

Пример 2

```
int n;
.  .  .
cin>>n;
int *mas=new int[n];
.  .  .
delete [] mas;
.  .  .
```

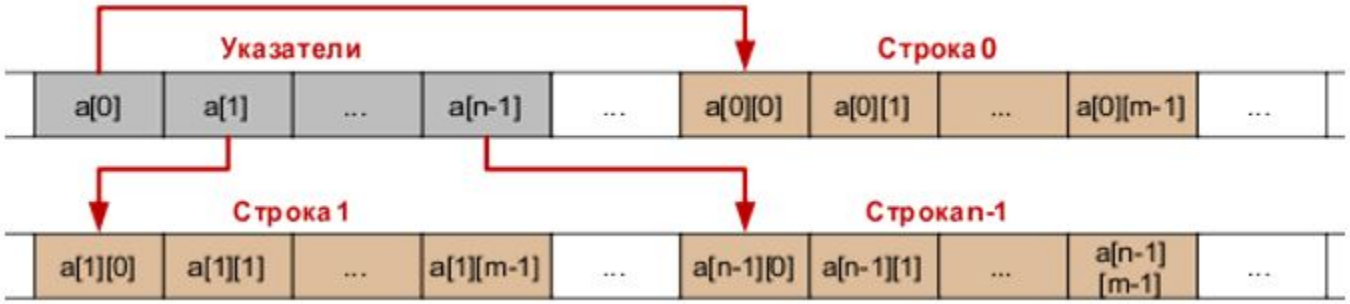
Двумерные динамические массивы

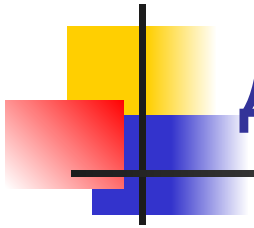
Динамическая матрица a(M,N)

```
int M, N ;  // M – number of rows, N – number of columns
int i, j;
cin >> M;
cin >> N;

int* a;
a=new int [M*N];
for ( i = 0; i < M; i++)          // input
    for ( j = 0; j < N; j++)
   cin >> a[i*N+j];         // *(a+i*N+j)=a[i*N+j]=a(i,j)
.     .     .
delete [] a;
```

Двумерные динамические массивы

```
int m, n ;  // m – число строк, n – число столбцов
int i, j;
cin >> m;
cin >> n;

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





Двумерные динамические массивы

```
for ( i = 0; i < m; i++)          // ввод массива
      for ( j = 0; j < n; j++)
      cin >> a[i][j];
  .     .     .

for ( i = 0; i < m; i++)
      delete [ ] a[i];            // освобождаем память, выделенную строкам
delete [ ] a;                     // освобождаем память, выделенную
                                  // указателям на строки
```