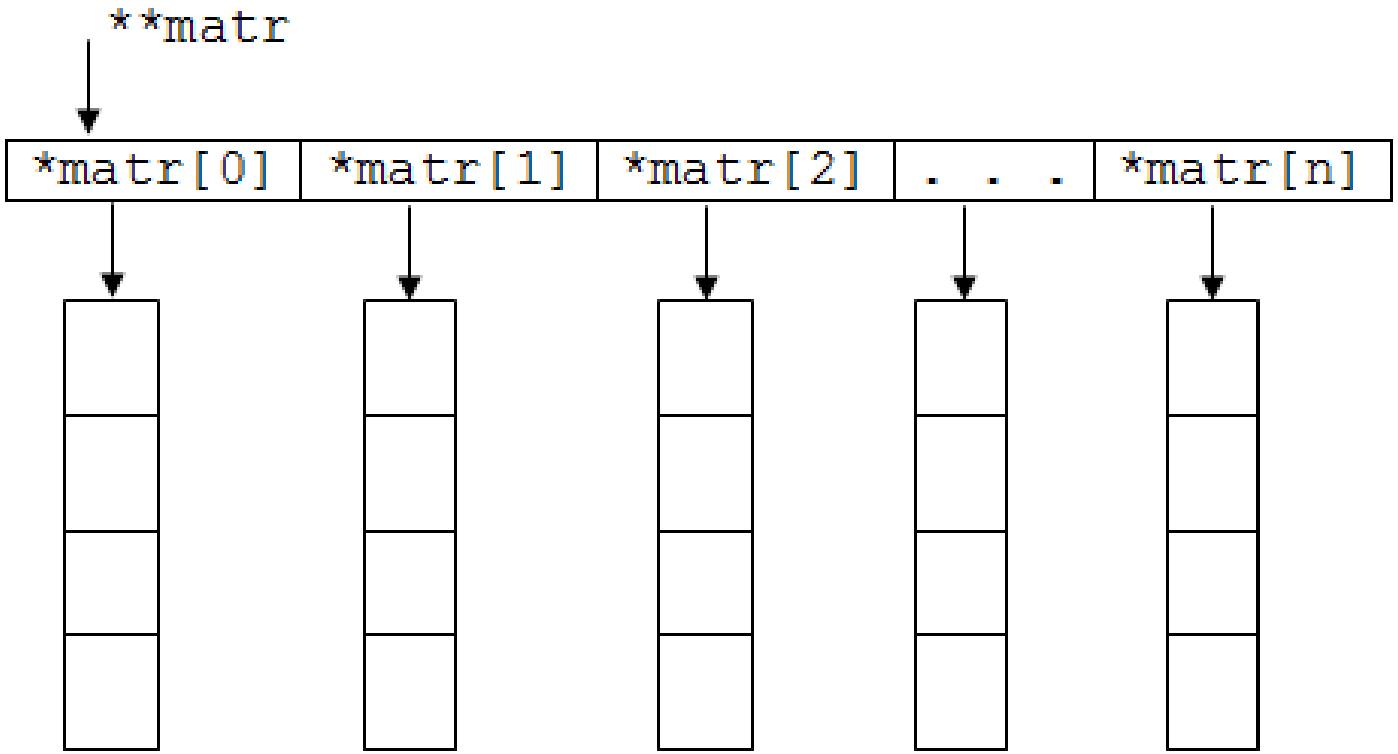
Лекция 12

Двумерные динамические массивы.
Массивы указателей.
Многомерные массивы.

Выделение памяти под двумерный динамический массив

При формировании двумерного динамического массива сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под одномерные массивы.



Многомерные массивы

При работе с динамической памятью в языке C++ существует ***2 способа*** выделения памяти под двумерный динамический массив:

1) *при помощи операции* **new**, которая позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать.

Синтаксис выделения памяти под массив указателей:

```
ИмяМассива = new Тип * [ВыражениеКонст];
```

Синтаксис выделения памяти для массива значений:

```
ИмяМассива[ЗначениеИндекса] = new Тип [ВыражениеКонст];
```

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип указателя на массив.

ВыражениеКонст – задает количество элементов (размерность) массива. Выражение *константного типа* вычисляется на этапе компиляции.

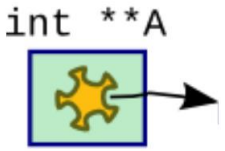
Многомерные массивы

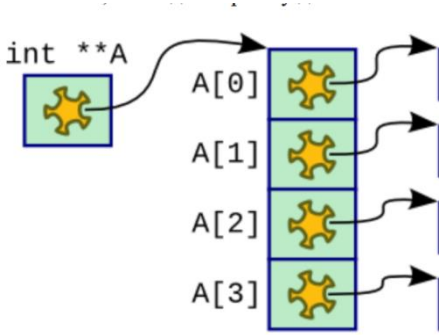
Пример:

```
int n=4, m=5;
```

//n и m – количество строк и столбцов матрицы

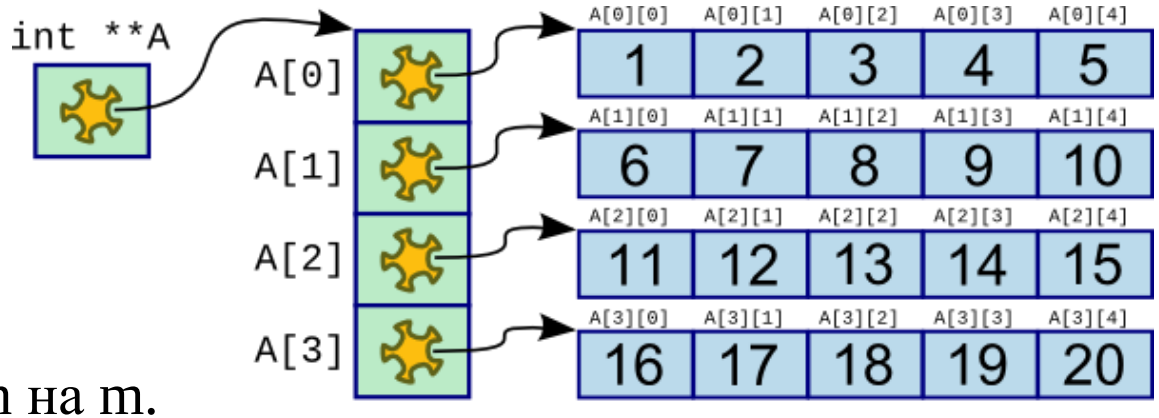
```
int **A;    //указатель на массив указателей
```






```
//выделение динамической памяти
//под массив указателей
A = new int * [n];

for (int i=0; i<n; i++)        //выделение памяти
   A[i] = new int [m];  // для массива значений
```



A[i][j] <=> *(*(A+i)+j)

В результате выделяем
динамическую память
под матрицу размером n на m.

Многомерные массивы

2) *при помощи библиотечной функции* **malloc** (**calloc**), которая предназначена для выделения динамической памяти.

Синтаксис выделения памяти под массив указателей:

```
ИмяМассива = (Тип **) malloc(N*sizeof(Тип *));
```

или

```
ИмяМассива = (Тип **) calloc(N, sizeof(Тип *));
```

Синтаксис выделения памяти для массива значений:

```
ИмяМассива[ЗначениеИндекса]=(Тип*)malloc(M*sizeof(Тип));
```

или

```
ИмяМассива[ЗначениеИндекса]=(Тип*)calloc(M, sizeof(Тип));
```

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип указателя на массив.

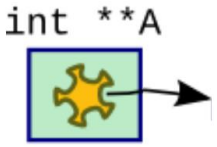
N – количество строк массива;

M – количество столбцов массива.

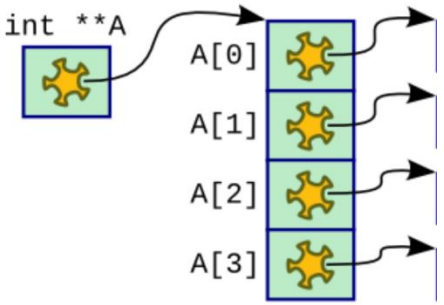
Многомерные массивы

Пример:

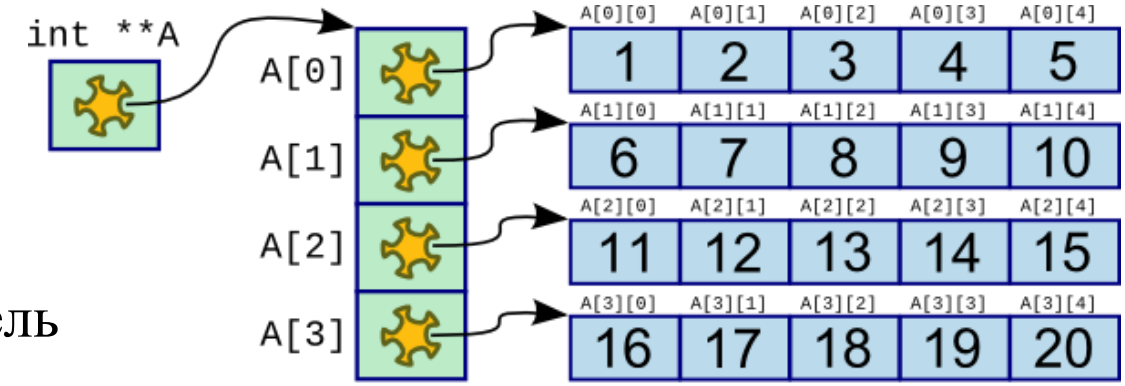
```
int n=4, m=5;  //n и m – количество строк и столбцов матрицы
int **A;  //указатель для массива указателей
```




```
//выделение памяти под массив указателей
A = (int **) malloc(n*sizeof(int *));
```




```
//выделение памяти для массива значений
for (int i=0; i<n; i++)
A[i] = (int *) malloc(m*sizeof(int));
```



A[i][j] <=> *(*(A+i)+j)

Так как функция **malloc (calloc)** возвращает нетипизированный указатель **void ***, то необходимо выполнять его преобразование в указатель объявленного типа.

Освобождение памяти, выделенной под двумерный динамический массив

Удаление из динамической памяти двумерного массива осуществляется в порядке, обратном его созданию, то есть сначала освобождается память, выделенная под одномерные массивы с данными, а затем память, выделенная под одномерные массив указателей.

Освобождение памяти, выделенной под двумерный динамический массив, также осуществляется 2 способами.

1) *при помощи операции* **delete**, которая освобождает участок памяти ранее выделенной операцией **new.**

Синтаксис освобождения памяти, выделенной для массива значений:

```
delete [] ИмяМассива [ЗначениеИндекса];
```

Синтаксис освобождения памяти, выделенной под массив указателей:

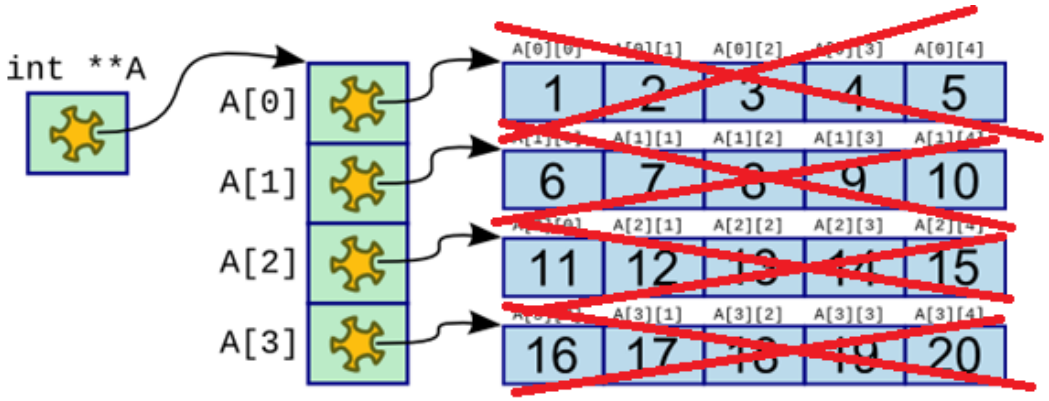
```
delete [] ИмяМассива;
```

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

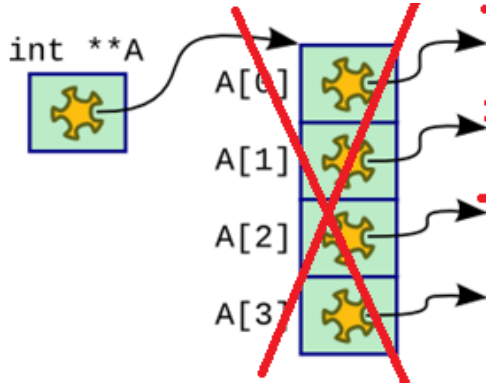
Освобождение памяти, выделенной под двумерный динамический массив

Пример:

```
for (int i=0; i<n; i++)
  delete [] A[i]; //освобождает память, выделенную для массива значений
```




```
delete [] A; //освобождает память, выделенную под массив указателей
```



Квадратные скобки [] означают,
что освобождается память, занятая
всеми элементами массива,
а не только первым.

Освобождение памяти, выделенной под двумерный динамический массив

2) *при помощи библиотечной функции* **free**, которая предназначена для освобождения динамической памяти.

Синтаксис освобождения памяти, выделенной для массива значений:

```
free (ИмяМассива[ЗначениеИндекса]);
```

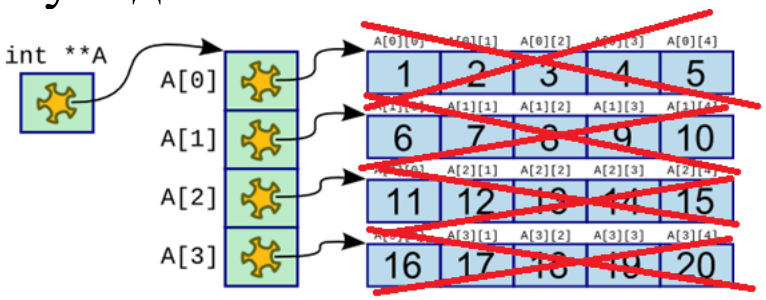
Синтаксис освобождения памяти, выделенной под массив указателей:

```
free (ИмяМассива);
```

Пример:

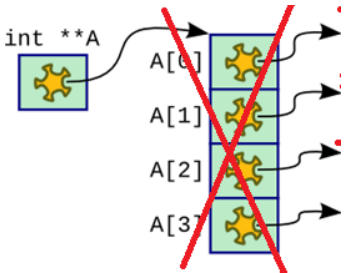
//освобождает память, выделенную для массива значений

```
for (int i=0; i<n; i++)
        free (A[i]);
```



//освобождает память, выделенную под массив указателей

```
free (A);
```



Управление памятью

```
#include <stdlib.h>
#include <iostream>
#include <windows.h>
using namespace std;
// объявления функций
void input_matr(int **a, int n, int m);
void output_matr(int **a, int n, int m);
int summa(int **a, int n, int m);

main()
{ int n, m, i;
   SetConsoleCP(65001); SetConsoleOutputCP(65001);

cout<<"Введите количество элементов: "; cin>>n>>m;
int **a= new int *[n];  //выделение памяти под массив из int*
   for (i=0; i<n; i++) a[i] = new int [m];
      //выделение памяти под массив из float
/*  float **b = (float **) malloc(n*sizeof(float *));
     for (i=0; i<n; i++)
          b[i] = (float *) malloc(m*sizeof(float)); */
```

Управление памятью

```
 input_matr(a, n, m); //   заполнение массива
   output_matr(a, n, m); //   печать массива
    cout<< "Сумма ="<< summa(a, n, m); //  нахождение суммы
 for (i=0; i<n; i++) delete [] a[i]; //   освобождение памяти
           delete [] a;
/* for (i=0; i<n; i++) free (b[i]); //   освобождение памяти
             free (b);*/
    return 0;
}
//------------- функция заполнения массива ----------
void input_matr(int **a, int n, int m)
{
int i,j;
  cout<<"Введите элементы массива: ";
   for (i=0; i<n; i++)
     for (j=0; j<m; j++)
        cin>>a[i][j];
   return;
}
```

Управление памятью

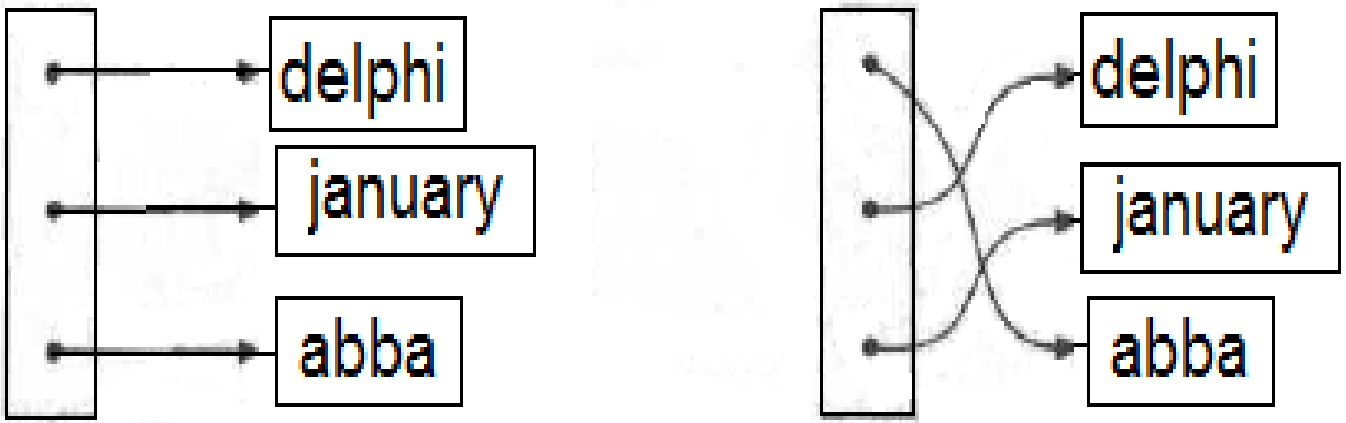
```
//----------- функция печати элементов массива
void output_matr(int **a, int n, int m)
{ int i,j;
  cout<<" Элементы массива"<<endl;
   for (i=0; i<n; i++)
    {     for (j=0; j<m; j++)  cout<<" "<<a[i][j];
           cout<<endl;}
   return;
}
//----------- нахождение суммы элементов массива
int summa(int **a, int n, int m)
{ int i, j, s=0;
   for (i=0; i<n; i++)
     for (j=0; j<m; j++)
          s+=a[i][j];
   return s;
}
```

```
Введите количество элементов: 2 3
Введите элементы массива: 1
2
3
-3
-2
-1
 Элементы массива
 1 2 3
 -3 -2 -1
Сумма =0
--------------------------------
```

Массивы указателей

Напишем программу, сортирующую в алфавитном порядке текстовые строки разной длины.

Воспользуемся **массивом указателей** на начала строк. Одна из возможностей сравнить две строки – передать указатели на них функции **`strcmp`.** Чтобы поменять местами строки, достаточно будет поменять местами в массиве их указатели (а не сами строки).



Массивы указателей

Процесс сортировки распадается на три этапа:

- ***чтение всех строк из ввода;***
- ***сортировка введенных строк;***
- ***печать их по порядку.***

Определим функции, соответствующие подзадачам, и напишем главную функцию **main**, управляющую этими функциями.

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#define MAXLINES 5000                 // максимальное число строк
char *lineptr[MAXLINES];              // массив указателей

int readlines(char *lineptr[ ], int nlines);     //ввод строк
void writelines(char *lineptr[ ], int nlines);   //печать строк
void qsort(char *lineptr[ ], int left, int right);//сортировка
```

Массивы указателей

```
/* сортировка строк */
int main( ) {
    SetConsoleCP(65001);
    SetConsoleOutputCP(65001);
   int nlines;    /* количество прочитанных строк */
   if ((nlines = readlines(lineptr, MAXLINES)) >= 0){
      qsort(lineptr, 0, nlines-1);
      writelines(lineptr, nlines);
      return 0;
   } else {
      printf("Ошибка: слишком много строк\n");
      return -1;
       }
}
```

Массивы указателей

```
#define MAXLEN 1000       /* максимальная длина строки */
int getline(char *, int);
char *alloc(int);
int readlines(char *lineptr[ ], int maxlines) {
   int len, nlines;
   char *p, line[MAXLEN];
   nlines = 0;
   while ((len = getline(line, MAXLEN)) > 0)
     if (nlines >= maxlines || (p = alloc(len)) == NULL)
      return -1;
     else { line[len-1] = '\0'; /* убираем символ \n */
          strcpy(p, line);
          lineptr[nlines++] = p;
        }
   return nlines;
}
```

Массивы указателей

```
/* writelines: печать строк  вариант 1*/
void writelines(char *lineptr[ ], int nlines) {
   int i;
   for (i = 0; i < nlines; i++)
      printf("%s\n", lineptr[i]);
}
```

```
/* writelines: печать строк вариант 2 */
void writelines(char *lineptr[], int nlines) {
   while (nlines-- > 0)
      printf( "%s\n", *lineptr++);
}
```

Вначале *** lineptr** указывает на первую строку; каждое приращение указателя приводит к тому, что ***lineptr** указывает на следующую строку, и делается это до тех пор, пока **nlines** не станет нулем.

Массивы указателей

```
/* qsort: сортирует v[left]...v[right] по возрастанию */
void qsort(char *v[ ], int left, int right) {
  int i, last;
  void swap(char *v[ ], int i, int j);
  if (left >= right)    /* ничего не делать, если в массиве */
      return;           /* менее двух элементов */
  swap(v, left, (left+ right)/2);
  last = left;
  for (i = left+1; i <= right; i++)
       if (strcmp(v[i], v[left]) < 0)
                swap(v, ++last, i);
  swap(v, left, last);
  qsort(v, left, last-1);
  qsort(v, last+1, right);
}
```

Массивы указателей

Небольшие поправки требуются и в функции перестановки.

```
/* swap: поменять местами v[i] и v[j] */
void swap(char *v[ ], int i, int j) {
  char *temp;
  temp = v[i];
  v[i] = v[j];
  v[j] = temp;
}
```

```
 /*выделение памяти под строку*/
 char *alloc(int n) {
 return (char *) malloc(n*sizeof(char));
 }
```

Массивы указателей

```
int getline(char *s, int lim){
   int c,i;
      for(i=0; i<lim-2; i++){
         c=getchar();
         if (c==EOF) return -1;
         if (c=='\n') break;
         s[i]=c;
          }
   s[i]='\0';
   i++;
   return i;
}
```

```
Севастополь
Керчь
Москва
Омск
Тюмень
Владивосток
Иваново
Йошкар-Ола
Алушта
^Z
Алушта
Владивосток
Иваново
Йошкар-Ола
Керчь
Москва
Омск
Севастополь
Тюмень
```

Многомерные массивы

Многомерные статические массивы задаются указанием каждого измерения в квадратных скобках:

```
int matr[3][5];
```

В памяти такой массив располагается построчно.

Массив **инициализируется** списком начальных значений, заключенным в фигурные скобки; каждая строка двумерного массива инициализируется соответствующим подсписком.

```
int daytab[2][13] = {
{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
{0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31} };
```

Нулевой столбец добавлен в начало **daytab** лишь для того, чтобы индексы, которыми мы будем пользоваться, совпадали с естественными номерами месяцев от 1 до 12.

Многомерные массивы

Строго говоря, в Си двумерный массив рассматривается как одномерный массив, каждый элемент которого – также массив. Индексирование изображается так:

```
daytab[i][j]        /* [строка] [столбец] */
```

а не так:

```
daytab[i,j]         /* НЕВЕРНО */
```

Если двумерный массив **передается функции** в качестве аргумента, то объявление соответствующего ему параметра **должно** содержать **кол-во столбцов, кол-во строк несущественно.**

Многомерные массивы

Таким образом, если массив **daytab** передается некоторой функции **f**, то эту функцию можно было бы определить так:

```
f(int daytab[2][13]) { . . . }
```

или

```
f(int daytab[ ][13]) { . . . }
```

или

```
f(int (*daytab)[13]) { . . . }
```

Последняя запись объявляет, что параметр есть указатель на массив из 13 значений типа **int**.

Многомерные массивы

В чем разница между двумерным массивом и массивом указателей?

Для двух следующих определений:

```
int a[10][20];
int *b[10];
```

записи **a[3][4]** и **b[3][4]** будут синтаксически правильным обращением к некоторому значению типа **int**.

Однако только **a** является истинно **двумерным массивом**: для двухсот элементов типа **int** будет выделена память, а вычисление смещения элемента **a[строка][столбец]** от начала массива будет вестись по формуле **20 x *строка* + *столбец***.

Для **b** же определено только 10 указателей, **причем без инициализации.** Предположим, что каждый элемент **b** указывает на двадцатиэлементный массив, в результате где-то будет выделено пространство, в котором разместятся 200 значений типа **int,** и еще 10 ячеек для указателей. Важное преимущество массива указателей в том, что **строки такого массива могут иметь разные длины**.

Т.е., каждый элемент массива **b** не обязательно указывает на двадцатиэлементный вектор; один может указывать на два элемента, другой – на пятьдесят, а некоторые и вовсе могут ни на что не указывать.

Многомерные массивы

Сравните определение массива указателей с объявлением для двумерного массива:

```
char aname[ ][15] = {"Неправ. месяц", "Янв", "Февр",
"Март"};
char *name[ ] = {"Неправильный месяц", "Янв", "Февр",
"Март"};
```