УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию в своей рабочей тетради, законспектируйте основные тезисы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: *mr.vickror@inbox.ru* до 17.04.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция № 14

Тема: «Многомерные массивы в языке С++. Массивы и символы указателей в языке С++»

План лекции:

1. **Многомерные массивы в языке С++.**
2. **Массивы и символы указателей в языке С++.**

Многомерные массивы

Массив массивов называется **многомерным массивом**:

```
int array[2][4]; // 2-элементный массив из 4-элементных массивов
```

Поскольку у нас есть 2 индекса, то это **двумерный массив**.

В двумерном массиве первый (левый) индекс принято читать как количество строк, а второй (правый) как количество столбцов. Массив выше можно представить следующим образом:

```
[0][0] [0][1] [0][2] [0][3] // строка №0
[1][0] [1][1] [1][2] [1][3] // строка №1
```

Чтобы получить доступ к элементам двумерного массива — просто используйте два индекса:

array[1][3] = 7; // без приставки int (типа данных)

Инициализация двумерных массивов

Для инициализации двумерного массива проще всего использовать вложенные фигурные скобки, где каждый набор значений соответствует определённой строке:

```
int array[3][5] =
{
{ 1, 2, 3, 4, 5 }, // строка №0
{ 6, 7, 8, 9, 10 }, // строка №1
{ 11, 12, 13, 14, 15 } // строка №2
};
```

Хотя некоторые компиляторы могут позволить вам упустить внутренние фигурные скобки, всё же рекомендуется указывать их в любом случае: улучшается читабельность + из-за того, что C++ заменяет отсутствующие инициализаторы значением 0:

```
int array[3][5] =
{
{ 2, 4  }, // строка №0 = 2, 4, 0, 0, 0
{ 1, 3, 7 }, // строка №1 = 1, 3, 7, 0, 0
{ 8, 9, 11, 12 } // строка №2 = 8, 9, 11, 12, 0
};
```

В двумерном массиве со **списком инициализаторов** можно не указывать только левый индекс (длину массива):

```
int array[][5] =
{
{ 1, 2, 3, 4, 5 },
{ 6, 7, 8, 9, 10 },
{ 11, 12, 13, 14, 15 }
};
```

Компилятор может сам вычислить количество строк в массиве. Однако не указывать два индекса — это уже ошибка:

```
int array[][] =
{
{ 3, 4, 7, 8 },
{ 1, 2, 6, 9 }
};
```

Подобно обычным массивам, многомерные массивы можно инициализировать значением 0 следующим образом:

```
int array[3][5] = { 0 };
```

Обратите внимание, это работает только в том случае, если вы явно объявляете длину массива (указываете левый индекс)! В противном случае, вы получите двумерный массив с 1 строкой.

Доступ к элементам в двумерном массиве

Для доступа ко всем элементам двумерного массива требуется два цикла: один для строк и один для столбцов. Поскольку доступ к двумерным массивам обычно выполняется по строкам, то левый индекс используется в качестве внешнего цикла:

```
for (int row = 0; row < numRows; ++row) // доступ по строкам
    for (int col = 0; col < numCols; ++col) // доступ к каждому элементу в строке
        std::cout << array[row][col];
```

Многомерные массивы больше двух измерений

Многомерные массивы могут быть больше двух измерений. Например, объявление трёхмерного массива:

int array[4][3][2];

Трёхмерные массивы трудно инициализировать любым интуитивным способом с использованием списка инициализаторов, поэтому лучше инициализировать весь массив значением 0 и явно присваивать значения с помощью вложенных циклов.

Доступ к элементам трёхмерного массива осуществляется так же, как и к элементам двумерного массива:

std::cout << array[3][2][1];

Пример двумерного массива

Рассмотрим пример использования двумерного массива:

```cpp
#include <iostream>

int main()
{
    // Объявляем массив 10x10
    const int numRows = 10;
    const int numCols = 10;
    int product[numRows][numCols] = { 0 };

    // Создаём таблицу умножения
    for (int row = 0; row < numRows; ++row)
        for (int col = 0; col < numCols; ++col)
            product[row][col] = row * col;

    // Выводим таблицу умножения
    for (int row = 1; row < numRows; ++row)
    {
        for (int col = 1; col < numCols; ++col)
            std::cout << product[row][col] << "\t";

        std::cout << '\n';
    }

    return 0;
}
```

Эта программа вычисляет и выводит таблицу умножения от 1 до 9 (включительно). Обратите внимание, при выводе таблицы в **цикле for** мы начинаем с 1 вместо 0. Это делается с целью предотвращения вывода нулевой строки с нулевыми столбцами, дабы в результате у нас не было строки с одними нулями!

Результат выполнения программы выше:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Двумерные массивы обычно используются в играх типа *tile-based*, где каждый элемент массива представляет собой один фрагмент/плитку. Они также используются в компьютерной 3D графике (в виде *матриц*) для вращения, масштабирования и отражения фигур.

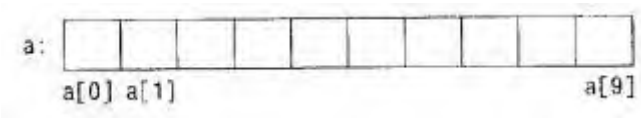
Массивы указателей

В Си существует связь между указателями и массивами, и связь эта настолько тесная, что эти средства лучше рассматривать вместе. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. Вариант с указателями в общем случае работает быстрее, но разобраться в нем, особенно непосвященному, довольно трудно.

Объявление

int a[10];

Определяет массив a размера 10, т. е. блок из 10 последовательных объектов с именами a[0], a[1], ..., a[9].



Запись a[i] отсылает нас к i-му элементу массива. Если pa есть указатель на int, т. е. объявлен как

int *pa;

то в результате присваивания

pa = &a[0];

pa будет указывать на нулевой элемент a, иначе говоря, pa будет содержать адрес элемента a[0].

Теперь присваивание

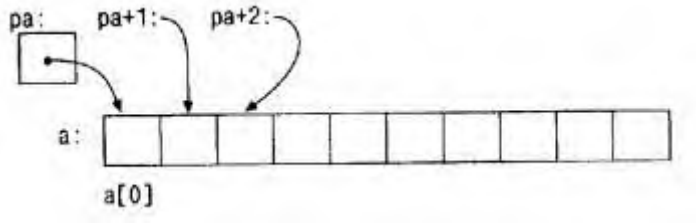
x = *pa;

будет копировать содержимое a[0] в x.

Если pa указывает на некоторый элемент массива, то pa+1 по определению указывает на следующий элемент, pa+i - на i-й элемент после pa, а pa-i - на i-й элемент перед pa. Таким образом, если pa указывает на a[0], то

*(pa+1)

есть содержимое a[1], a+i - адрес a[i], а *(pa+i) - содержимое a[i].



Сделанные замечания верны безотносительно к типу и размеру элементов массива a. Смысл слов "добавить 1 к указателю", как и смысл любой арифметики с указателями, состоит в том, чтобы pa+1 указывал на следующий объект, а pa+i - на i-й после pa.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. После присваивания

pa = &a[0];

pa и a имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание pa=&a[0] можно также записать в следующем виде:

pa = a;

Еще более удивительно (по крайней мере на первый взгляд) то, что a[i] можно записать как *(a+i). Вычисляя a[i], Си сразу преобразует его в *(a+i); указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора & записи &a[i] и a+i также будут эквивалентными, т. е. и в том и в другом случае это адрес i-го элемента после a. С другой стороны, если pa - указатель, то его можно использовать с индексом, т. е. запись pa[i] эквивалентна записи *(pa+i). Короче говоря,

элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель - это переменная, поэтому можно написать pa=a или pa++. Но имя массива не является переменной, и записи вроде a=pa или a++ не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться отмеченным фактом и написать еще одну версию функции strlen, вычисляющей длину строки.

```
/* strlen: возвращает длину строки */
int strlen(char *s)
{
   int n;
   for (n = 0; *s != '\0'; s++)
      n++;
   return n;
}
```

Так как переменная s - указатель, к ней применима операция ++; s++ не оказывает никакого влияния на строку символов функции, которая обратилась к strlen. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции strlen. Это значит, что все вызовы, такие как:

```
strlen("Здравствуй, мир"); /* строковая константа */
strlen(array);          /* char array[100]; */
strlen(ptr);            /* char *ptr; */
```

правомерны.

Формальные параметры

```
char s[];
```

и

*char *s;*

в определении функции эквивалентны. Мы отдаем предпочтение последнему варианту, поскольку он более явно сообщает, что s есть указатель. Если функции в качестве аргумента передается имя массива, то она может рассматривать его так, как ей удобно - либо как имя массива, либо как указатель, и поступать с ним соответственно. Она может даже использовать оба вида записи, если это покажется уместным и понятным.

Функции можно передать часть массива, для этого аргумент должен указывать на начало подмассива. Например, если a - массив, то в записях

f(&a[2])

или

f(a+2)

функции f передается адрес подмассива, начинающегося с элемента a[2]. Внутри функции f описание параметров может выглядеть как

f(int arr[]) {...}

или

f(int *arr) {...}

Следовательно, для f тот факт, что параметр указывает на часть массива, а не на весь массив, не имеет значения.

Если есть уверенность, что элементы массива существуют, то возможно индексирование и в "обратную" сторону по отношению к нулевому элементу; выражения p[-1], p[-2] и т.д. не противоречат синтаксису языка и обращаются к элементам, стоящим непосредственно перед p[0]. Разумеется, нельзя "выходить" за границы массива и тем самым обращаться к несуществующим объектам.

Контрольные вопросы:

1. **Привести пример многомерного массива.**
2. **Привести пример многомерного массива с указателем.**