Урок №93. Указатели на указатели

Юрий |

- Уроки C++

|

Обновл. 16 Авг 2020 |

Указатель на указатель — это именно то, что вы подумали: **указатель**, который содержит адрес другого указателя.

Оглавление:

Указатели на указатели

Обычный указатель типа int объявляется с использованием одной звёздочки:

```
1 int *ptr; // указатель типа int, одна звёздочка
```

Указатель на указатель типа int объявляется с использованием 2-х звёздочек:

```
1 int **ptrptr; // указатель на указатель типа int (две звёздочки)
```

Указатель на указатель работает подобно обычному указателю: вы можете его разыменовать для получения значения, на которое он указывает. И, поскольку этим значением является другой указатель, для получения исходного значения вам потребуется выполнить разыменование еще раз. Их следует выполнять последовательно:

```
#include <iostream>

int main()
{
    int value = 7;

    int *ptr = &value;
    std::cout << *ptr << std::endl; // разыменовываем указатель, чтобы получить значени

    int **ptrptr = &ptr;
    std::cout << **ptrptr << std::endl;

    return 0;
}
```

Результат выполнения программы:

```
7
7
```

Обратите внимание, вы не можете инициализировать указатель на указатель напрямую значением:

```
int value = 7;
int **ptrptr = &&value; // нельзя
```

Это связано с тем, что оператор адреса (&) требует **l-value**, но `&value` — это r-value. Однако указателю на указатель можно задать **значение null**:

```
int **ptrptr = nullptr; // используйте 0, если не поддерживается C++11
```

Массивы указателей

Указатели на указатели имеют несколько применений. Наиболее используемым является **динамическое выделение** массива указателей:

```
int **array = new int*[20]; // выделяем массив из 20 указателей типа int
```

Это тот же обычный **динамически выделенный массив**, за исключением того, что элементами являются указатели на тип int, а не значения типа int.

Двумерные динамически выделенные массивы

Другим распространенным применением указателей на указатели является динамическое выделение **многомерных массивов**. В отличие от двумерного фиксированного массива, который можно легко объявить следующим образом:

```
int array[15][7];
```

Динамическое выделение двумерного массива немного отличается. У вас может возникнуть соблазн написать что-то вроде следующего:

```
int **array = new int[15][7]; // не будет работать!
```

Здесь вы получите ошибку. Есть два возможных решения. Если правый индекс является **константой типа compile-time**, то вы можете сделать следующее:

```
int (*array)[7] = new int[15][7];
```

Скобки здесь потребуются для соблюдения **приоритета**. В C++11 хорошей идеей будет использовать **ключевое слово auto** для автоматического определения типа данных:

```
auto array = new int[15][7]; // намного проще!
```

К сожалению, это относительно простое решение не работает, если правый индекс не является константой типа compile-time. В таком случае всё немного усложняется. Сначала мы выделяем массив указателей (как в примере, приведенном выше), а затем перебираем каждый элемент массива указателей и выделяем динамический массив для каждого элемента этого массива. Итого, наш динамический двумерный массив — это динамический одномерный массив динамических одномерных массивов!

```
int **array = new int*[15]; // выделяем массив из 15 указателей типа int – это наши стр
for (int count = 0; count < 15; ++count)
    array[count] = new int[7]; // а это наши столбцы
```

Доступ к элементам массива выполняется как обычно:

```
array[8][3] = 4; // это то же самое, что и (array[8])[3] = 4;
```

Этим методом, поскольку каждый столбец массива динамически выделяется независимо, можно сделать динамически выделенные двумерные массивы, которые не являются прямоугольными. Например, мы можем создать массив треугольной формы:

```
int **array = new int*[15]; // выделяем массив из 15 указателей типа int – это наши стр
for (int count = 0; count < 15; ++count)
    array[count] = new int[count+1]; // а это наши столбцы
```

В примере, приведенном выше, `array[0]` — это массив длиной 1, а `array[1]` — массив длиной 2 и т.д.

Для освобождения памяти динамически выделенного двумерного массива (который создавался с помощью этого способа) также потребуется цикл:

```
for (int count = 0; count < 15; ++count)
    delete[] array[count];
delete[] array; // это следует выполнять в конце
```

Обратите внимание, мы удаляем массив в порядке, противоположном его созданию. Если мы удалим массив перед удалением элементов массива, то нам придется получать доступ к освобожденной памяти для удаления элементов массива. А это, в свою очередь, приведет к неожиданным результатам.

Поскольку процесс выделения и освобождения двумерных массивов является несколько запутанным (можно легко наделать ошибок), то часто проще «сплющить» двумерный массив в одномерный массив:

```
// Вместо следующего:
int **array = new int*[15]; // выделяем массив из 15 указателей типа int – это наши стр
for (int count = 0; count < 15; ++count)
    array[count] = new int[7]; // а это наши столбцы

// Делаем следующее:
int *array = new int[105]; // двумерный массив 15x7 "сплющенный" в одномерный массив
```

Простая математика используется для конвертации индексов строки и столбца прямоугольного двумерного массива в один индекс одномерного массива:

```
int getSingleIndex(int row, int col, int numberOfColumnsInArray)
{
     return (row * numberOfColumnsInArray) + col;
}

// Присваиваем array[9,4] значение 3, используя наш "сплющенный" массив
array[getSingleIndex(9, 4, 5)] = 3;
```

Указатель на указатель на указатель на указатель и т.д.

Также можно объявить указатель на указатель на указатель:

```
int ***ptrx3;
```

Они могут использоваться для динамического выделения трехмерного массива. Тем не менее, для этого потребуется цикл внутри цикла и чрезвычайная аккуратность и осторожность, чтобы не наделать ошибок. Вы даже можете объявить указатель на указатель на указатель на указатель:

```
int ****ptrx4;
```

Или сделать еще большую вложенность, если захотите. Однако на практике такие указатели редко используются.

Заключение

Рекомендуется применять указатели на указатели только в самых крайних случаях, так как они сложны в использовании и потенциально опасны. Достаточно легко разыменовать нулевой или «висячий» указатель в ситуациях с использованием обычных указателей, вдвое легче это сделать в ситуациях с указателем на указатель, поскольку для получения исходного значения потребуется выполнить двойное разыменование!

Оценить статью:

(**197** оценок, среднее: **4,90** из 5)

Урок №92. Указатели типа void

Урок №94. Введение в std::array

Комментариев: 22

1.  *Юрий*:
[21 ноября 2020 в 17:44](#)

Почему в функции освобождении памяти компилятор ругается, что "Использование неинициализированной памяти " на delete[]parray[i]?

```
#include <iostream>
#include<ctime>
using namespace std;
int** arrayMem(int** parray, int row, int col);
void fillArray(int** parray, int row, int col);
void drawArray(int** parray, int row, int col);
void drawArray(int** parray, int row);
int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "rus");
    int row = 0;
    int col = 0;
    int** array = 0;
    cout << "Количество строк:\n";
    cin >> row;
    cout << "количество столбцов:\n";
    cin >> col;
    array = arrayMem(array, row, col);
    fillArray(array, row, col);
    drawArray(array, row, col);
    return 0;
}

int** arrayMem(int** parray, int row, int col)
{
    parray = new int* [row];
    for (int i = 0; i < row; i++)
```

```
29          parray[i] = new int[col];
30      return parray;
31  }
32  void fillArray(int** parray, int row, int col)
33  {
34      for (int i = 0; i < row; i++)
35      {
36          for (int j = 0; j < col; j++)
37          {
38              parray[i][j] = 10 + rand() % 41;
39          }
40      }
41  }
42
43  void drawArray(int** parray, int row, int col)
44  {
45      for (int i = 0; i < row; i++)
46      {
47          for (int j = 0; j < col; j++)
48          {
49              cout << parray[i][j] << " | ";
50          }cout << endl;
51      }
52  }
53  void drawArray(int** parray, int row)
54  {
55      for (int i = 0; i < row; i++)
56      {
57          delete[]parray[i];
58          delete[]parray;
59      }
60
61  }
```

Ответить

1.  *Spardoks*:
   30 ноября 2020 в 17:37

```
1  void drawArray(int** parray, int row)
2  {
3      for (int i = 0; i < row; i++)
4      {
5          delete[]parray[i];
6          delete[]parray;
7      }
8
9  }
```

Здесь в delete[] parray лишнее. За цикл нужно вынести, чтоб освободить память только 1 раз

Ответить

2.  *Данил*:
15 июня 2020 в 18:26

В параметрах функции мне необходимо сделать константный указатель на указатель(т.к. функция только выводит двумерный массив), но тогда при ее вызове компилятор ругается на то, что невозможно преобразовать аргумент из "double **" в "const double **. Почему так происходит?
Вот сама функция:

```
void PrintArray(double **array, int N) {
    for (int y = 0; y < N; ++y) {
        for (int x = 0; x < N; ++x) {
            std::cout << array[y][x];
            std::cout << " ";
            if (x == N - 1)std::cout << "\n";
        }
    }
}
```

Вот ее вызов:

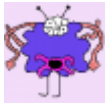
```
PrintArray(arrayEnd, N);
```

Ответить

1.  *Andrey*:
6 октября 2020 в 02:29

```
void PrintArray(double *const*const array, int N) {
    for (int y = 0; y < N; ++y) {
        for (int x = 0; x < N; ++x) {
            std::cout << array[y][x];
            std::cout << " ";
            if (x == N - 1)std::cout << "\n";
        }
    }
}
```

Ответить

3.  *Nicolai*:
1 июня 2020 в 18:15

Паровозик ptrptrptr….

Ответить

1.  *Сергей*:
[11 октября 2020 в 23:00](#)

После прочтения этой темы понимаю, что мозг возмущенно отказывается переваривать эти указатели на указатели массивов и тут твой комментарий, поржал)

Ответить

4.  *alex_1988*:
[11 февраля 2020 в 21:35](#)

Накопившиеся вопросы:

1) Почему при передачи в функции двухмерных массивов типы указателей разнятся?
пример:

```
void staticArray(int mass[2][3])  // подходит для фиксированого
void dinamicArray(int** mass)  // подходит для динамического
```

2) Как правильно почистить переменную через delete чтобы не произошла утечка памяти в этом коде:

```
int** ptr = new int*;   // динамический указатель на указатель
*ptr = new int(10);
cout << **ptr << "\n";
```

Ответить

1.  *alex_1988*:
[15 февраля 2020 в 15:20](#)

Покумекал сам и думаю пришел к правильным выводам:

1) Фиксированный массив передается в функцию таким образом потому что является настоящим двухмерным массивом, где все данные хранятся в памяти поочередно и не содержут указатель на другие подмассивы как это сделано в динамическом.

2) Чтобы удалить динамический указатель на указатель без утечки памяти нужно сделать так:

```
delete (*ptr);
delete ptr;
```

Ответить

5.  *Артём*:
[29 октября 2019 в 23:04](#)

Объясните,пожалуйста,конкретнее,как происходит конвертации индексов строки и столбца прямоугольного двумерного массива в один индекс одномерного массива.

Ответить

1.  *Никита*:
   30 октября 2019 в 00:20

   Ну как одну строку на 100 символов, разбить на 5 строк по 20 символов понятно?

   Ответить

   1.  *Артём*:
      30 октября 2019 в 19:36

      Понятно. Я прошу объяснить на данном примере. Непонятно, что означают числа 9,4,5,что такое array[9,4] и как получается формула в return.

      Ответить

      1.  *Nikita*:
         3 ноября 2019 в 17:11

         Главное я тогда когда был на этом материале, тоже не понял, долго вникал, расписывал и разрисовывал себе этот массив , и всёравно понадобилось достаточно много времени чтобы наконец-то разобраться что к чему.
         Тут приходит уведомление на почту о твоём сообщении тут, я захожу, и опять не понимаю… И смешно и грустно одновременно))

         Но поднапрягшись таки вспомнил)
         Смотри. 9 и 4 это координаты элемента массива который нам нужен для присваивания 3ки (в случае выше)
         Число 5 — это количество столбцов в нашем массиве (не адрес элемента) Нужно нам количество столбцов чтобы умножить их на количество строк. Таким образом мы как бы восстанавливаем 2мерный массив из одномерного.
         Для этого мы умножаем 9 (строка на самом деле 10я, но отсчет с 0)
         на реальное количество столбцов (тут разумеется отсчёт с 1)) на 5.

         Получаем "индекс" 45 (9*5).
         Теперь осталось подобраться к самому элементу.
         Добавляем 4. Тем самым мы перепрыгиваем с координат(8,4 — адрес 45ого элемента для ПК(для нас, по человечески это 9я строка 5 столбец) и оказываемся уже на 10строке (для ПК 9я, ведь с 0 считаем)
         В общем по человечески это 50й элемент в массиве. Для ПК индекс 49.
         Знаю, мудрено очень обьяснил, но надеюсь хоть часть моей инфы, будет в помощь.

2.  *Nikita*:
[3 ноября 2019 в 17:18](#)

Дополнение:
То что в скобках, удаляем , написал неверно:
"Для этого мы умножаем 9 (строка на самом деле 10я, но отсчет с 0)"

строка на этот момент 9я для человека и 8я для ПК. После того как добавим 4ку, строка +1

6.  *Артём*:
[21 августа 2019 в 10:12](#)

```
int getSingleIndex(int row, int col, int numberOfColumnsInArray)
{
     return (row * numberOfColumnsInArray) + col;
}

// Присваиваем array[9,4] значение 3, используя наш "сплющенный" массив
array[getSingleIndex(9, 4, 5)] = 3;
```

Не совсем понял этот момент. Воссоздал массив на листочке, элемент [9,4] как будто является 50-ым, хотя по формуле он 49-ый. И что значит параметр "numberOfColumnsInArray"? Помогите, пожалуйста. Заранее спасибо!

[Ответить](#)

1.  *Артём*:
[21 августа 2019 в 19:35](#)

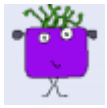
Всё. Разобрался(я просто затупил).

[Ответить](#)

1.  *Никита*:
[29 августа 2019 в 16:27](#)

Объясни пожалуйста. А то я остался на стадии тупняка. Массив у нас какой изначально ? 15*7?

[Ответить](#)

7.  *Владимир*:
[4 декабря 2018 в 13:47](#)

Оставлю это здесь. Может, кому интересно будет на это посмотреть:

```
#include <iostream>
#include <stdint.h>
#include <cstdlib>
#include <ctime>

using std::cout;

int main()
{
    srand(time(NULL));
    rand();

    const uint16_t size{ 2 };

    uint16_t *****x_5Ptr;
                                                        //Начинается самое
    x_5Ptr = new uint16_t****[size];                    //Выделено по учас

    for (uint16_t i = 0; i < size; ++i)
    {
        x_5Ptr[i] = new uint16_t***[size];              //Выделено по учас

        for (uint16_t j = 0; j < size; ++j)
        {
            x_5Ptr[i][j] = new uint16_t**[size];        //Выделено по учас

            for (uint16_t k = 0; k < size; ++k)
            {
                x_5Ptr[i][j][k] = new uint16_t*[size];  //Выделено по учас

                for (uint16_t l = 0; l < size; ++l)
                {
                    x_5Ptr[i][j][k][l] = new uint16_t[size];  //Выделено по учас
                }
            }
        }
    }

    //Итого ОС выделит нам 32 ячейки памяти по n * 32 байта (n = 4, если приложени

    uint16_t totalSizeOfx_5Ptr{};

    for (uint16_t a = 0; a < size; ++a)
        for (uint16_t b = 0; b < size; ++b)
            for (uint16_t c = 0; c < size; ++c)
                for (uint16_t d = 0; d < size; ++d)

```

```
                    for (uint16_t e = 0; e < size; ++e)
                    {
                        x_5Ptr[a][b][c][d][e] = rand() % 10;

                        totalSizeOfx_5Ptr += sizeof(x_5Ptr);
                    }

    cout << "Total adresses's size: " << totalSizeOfx_5Ptr << " bytes.\n\n";
    for (uint16_t a = 0; a < size; ++a)
    {
        for (uint16_t b = 0; b < size; ++b)
        {
            for (uint16_t c = 0; c < size; ++c)
            {
                for (uint16_t d = 0; d < size; ++d)
                {
                    for (uint16_t e = 0; e < size; ++e)
                    {
                        cout << a << b << c << d << e << " Value: " << x_5Ptr[a][
                    }
                    cout << '\n';
                }
                cout << '\n';
            }
            cout << '\n';
        }
        cout << '\n';
    }

    for (uint16_t i = 0; i < size; ++i)
    {
        for (uint16_t j = 0; j < size; ++j)
        {
            for (uint16_t k = 0; k < size; ++k)
            {
                for (uint16_t l = 0; l < size; ++l)
                {
                    delete[] x_5Ptr[i][j][k][l];      //Возвращаем в ОС все ячейки
                }
                delete[] x_5Ptr[i][j][k];             //Возвращаем в ОС все ячейки
            }
            delete[] x_5Ptr[i][j];                    //Возвращаем в ОС все ячейки
        }
        delete[] x_5Ptr[i];                           //Возвращаем в ОС все ячейки
    }
    delete[] x_5Ptr;                                  //Возвращаем в ОС все ячейки


```

```
    x_5Ptr = nullptr;                                   //Обнуление x_5Ptr;
    return 0;
}
```

Ответить

1.  *Алексей*:
   1 июня 2019 в 11:57

   Пятимерный массив… Пятимерное пространство… Хмм, это было неплохо)

   Ответить

   1.  *rutrud*:
      16 октября 2019 в 22:35

      спасибо, как раз призадумался какое отличие от С в создании и освобождении многомерных массивов.

      Ответить

8.  *Виталий*:
   17 апреля 2018 в 03:44

```
int (*array)[7] = new int[15][7];
```

зачем здесь второй индекс [7]??

Ответить

1.  *Юрий*:
   17 апреля 2018 в 22:49

   Здесь (*array) — это одномерный динамический массив, а [7] — это одномерный фиксированный массив. [7] нужен для создания двумерного динамического массива.

   Ответить

2.  *Михаил*:
   27 мая 2020 в 03:25

   Потому что вот это

```
int (*array)[7] = new int[15][7];
```

   на русском читается примерно так:
   1) выдели мне указатель на 7 интов.

2) а также выдели массив из 15 раз по 7 интов и присвой адрес первой семерки интов в мой указатель (с первого шага).

Итак, почему нужен указатель на 7 интов для этого.

Похоже разобрался с этим, может полезно будет кому.
Вот рассмотрим такой код, когда имеет двумерный массив представлен одномерным массивом указателей на одномерные массивы:

```
    int **arr = new int* [15];
    std::cout << arr << std::endl;
    for (int y = 0; y < 15; y++) {
        arr[y] = new int [7];
        std::cout << arr[y] << std::endl;
        for (int x = 0; x < 7; x++) {
            arr[y][x] = y*16 + x;
        }
    }<pre>
Фактически создается массив указателей на инт. В этом массиве сплошняком лежа

Теперь сравним с таким кодом:
<pre>   int (*arr)[7]= new int[15][7];
    std::cout << arr << std::endl;
    for (int y = 0; y < 15; y++) {
        std::cout << &arr[y] << std::endl;
        for (int x = 0; x < 7; x++) {
            arr[y][x] = y*16 + x;
        }
    }
```

Тут указатель на начало массива один. А весь массив (15*7 элементов) располагается в памяти спошняком — первый идет элемент [0][0], потом [0][1] и так до [0][7], потом идет [1][0] и т. д.
И когда мы делаем так

```
arr[y][x]
```

компилятор генерит для этого код, который арифметикой указателей вычисляет адрес нужного элемента двумерного массива:
arr+y*размер_внутреннего_измерения*sizeof(int)+ x*sizeof(int)
А откуда компилятору взять этот размер_внутреннего_измерения, поэтому нам приходится его указывать.

Ответить

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий

Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

Получать уведомления о новых комментариях по электронной почте. Вы можете подписаться без комментирования.

Отправить комментарий

TELEGRAM КАНАЛ

ПАБЛИК

ТОП СТАТЬИ

- Словарь программиста. Сленг, который должен знать каждый кодер
- Урок №1. Введение в программирование
- 70+ бесплатных ресурсов для изучения программирования
- Урок №1: Введение в создание игры «SameGame» на C++/MFC
- Урок №4. Установка IDE (Интегрированной Среды Разработки)