Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
Институт Информационных Технологий
Кафедра Промышленной Информатики

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Тема лекции «Динамические массивы»

Лектор Каширская Елизавета Натановна (к.т.н., доцент, ФГБОУ ВО "МИРЭА - Российский технологический университет") e-mail: liza.kashirskaya@gmail.com

Лекция 8

Введение

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера. Например, мы написали какую-то программку, которая обрабатывает массив. При написании данной программы необходимо было объявить массив, то есть задать ему фиксированный размер (к примеру, от 0 до 100 элементов). Тогда данная программа будет не универсальной, ведь может обрабатывать массив размером не более 100 элементов. А если нам понадобятся всего 20 элементов, но в памяти выделится место под 100 элементов, ведь объявление массива было статическим, а такое использование памяти крайне неэффективно.

Когда мы разбирали понятие массива, то при объявлении мы задавали массиву определенный постоянный размер. Размером статического массива должна являться числовая константа, а не переменная.

В большинстве случаев, целесообразно выделять определенное количество памяти для массива, значение которого изначально неизвестно, например, если необходимо создать динамический массив из **N** элементов, где значение **N** задается пользователем. Мы уже учились выделять память для переменных, используя указатели. Выделение памяти для динамического массива имеет аналогичный принцип.

Динамический массив — это массив с элементами, выделенными в динамической памяти. Он необходим в случае, если размер неизвестен на этапе компиляции, или если размер достаточно большой, и мы не хотим выделять массив на стеке, размер которого обычно сильно ограничен.

Динамические объекты обычно используются, когда невозможно привязать время жизни объекта к какой-то конкретной области видимости. Если это можно сделать наверняка следует использовать автоматическую память. Но это предмет отдельного рассмотрения.

Когда динамический объект создан, кто-то должен его удалить, и условно типы объектов можно разделить на две группы: те, которые никак не осведомлены о процессе своего удаления, и те, которые что-то подозревают. Будем говорить, что первые имеют стандартную модель управления памятью, а вторые — нестандартную.

К типам со стандартной моделью управления памятью относятся все стандартные типы.

Иными словами, тип со стандартной моделью управления памятью не предоставляет никаких дополнительных механизмов для управления своим временем жизни. Этим целиком и полностью должна заниматься пользовательская сторона.

ДИНАМИЧЕСКИЕ МАССИВЫ

Поскольку массивам фиксированного размера память выделяется во время компиляции, то здесь мы имеем два ограничения.

Первое ограничение. Массивы фиксированного размера не могут иметь длину, основанную на любом пользовательском вводе или другом значении, которое вычисляется во время выполнения программы (runtime).

Второе ограничение. Фиксированные массивы имеют фиксированную длину, которую нельзя изменить.

Во многих случаях эти ограничения являются проблематичными. К счастью, C++ поддерживает еще один тип массивов, известный как **динамический массив**. Размер такого массива может быть установлен во время выполнения программы, и его можно изменить.

ДИНАМИЧЕСКИЕ МАССИВЫ

Как уже было сказано — при объявлении статического массива, его размером должна являться числовая константа, а не переменная. В большинстве случаев целесообразно выделять определенное количество памяти для массива, значение которого изначально неизвестно.

Например, необходимо создать динамический массив из **N** элементов, где значение **N** задается пользователем. Мы учились выделять память для переменных, используя указатели. Выделение памяти для динамического массива имеет аналогичный принцип.

ДИНАМИЧЕСКИЕ МАССИВЫ

При объявлении массива мы задавали массиву определенный постоянный размер. Возможно, кто-то пробовал делать так:

```
int n=10;
int arr[n];
```

Казалось бы, это должно сработать, ведь размер массива равен **n** только в момент инициализации массива, а при изменении n во время работы программы размер массива не изменится.

Но компилятор это не пропустит, так как int n=10; - это переменная, а нужна константа:

```
const int n=10;
```

Создание динамического массива

Операция new создает объект заданного типа, выделяет ему память и возвращает указатель правильного типа на данный участок памяти. Если память невозможно выделить, например, в случае отсутствия свободных участков, то возвращается нулевой указатель, то есть указатель вернет значение 0. Выделение памяти возможно под любой тип данных: int, float, double, char и т.д.
Чаще всего операции new и delete применяются для создания динамических массивов, а не для создания динамических переменных.
Создание динамического массива покажем на примере.

Пример.
int *numbers = new int[4]; // динамический массив из четырех чисел

В этом случае оператор new также возвращает указатель на объект типа int - первый элемент в созданном массиве.
Приведенный пример определяет массив из четырех элементов типа int, но каждый из них имеет неопределенное значение.

Пример.

```
// пример использования операции new
int *ptrvalue = new int;
//где ptrvalue – указатель на выделенный участок памяти типа int
//new – операция выделения свободной памяти под создаваемый объект.
```

Использование операции new для создания динамических массивов

Предположим, например, что вы пишете программу, которой может понадобиться массив, а может, и нет — это зависит от информации, поступающий во время выполнения. Если вы создаете массив простым объявлением, пространство для него распределяется раз и навсегда — во время компиляции. Будет ли востребованным массив в программе или нет — он все равно существует и занимает место в памяти.

Пока что мы рассмотрим два важных обстоятельства относительно динамических массивов: как применять операцию new для создания массива и как использовать указатель для доступа к его элементам.

Как всегда, вы должны сбалансировать каждый вызов new соответствующим вызовом delete, когда программа завершает работу с этим блоком памяти. Однако использование new с квадратными скобками для создания массива требует применения альтернативной формы delete при освобождении массива:

delete [] psome; // *освобождение динамического массива*

Присутствие квадратных скобок сообщает операционной системе, что она должна освободить весь массив, а не только один элемент, на который указывает указатель. Обратите внимание, что скобки расположены между delete и указателем.

Указатели на массивы

Указатель — это переменная, хранящая в себе адрес ячейки оперативной памяти.

Мы можем обращаться, например, к массиву данных через указатель, который будет содержать адрес начала диапазона ячеек памяти, хранящих этот массив.

После того, как этот массив станет не нужен для выполнения остальной части программы, мы просто освободим память по адресу этого указателя, и она вновь станет доступна для других переменных.

Создание и удаление динамических массивов

Если вы используете new без скобок, то и соответствующая операция delete тоже должна быть без скобок. Если же new со скобками, то и соответствующая операция delete должна быть со скобками. Ранние версии C++ могут не распознавать нотацию с квадратными скобками. Согласно стандарту ANSI/ISO, однако, эффект от несоответствия формы new и delete не определен, т.е. вы не должны рассчитывать в этом случае на какое-то определенное поведение. Вот пример:

```
int * pt = new int;
short * ps = new short [500] ;
delete [] pt; // эффект не определен, не делайте так
delete ps; // эффект не определен, не делайте так
```

• Не использовать delete для освобождения той памяти, которая не была выделена new.

• Не использовать delete для освобождения одного и того же блока памяти дважды.

• Использовать delete [], если применялась операция new[] для размещения массива.

• Использовать delete без скобок, если применялась операция new для размещения отдельного элемента.

• Помнить о том, что применение delete к нулевому указателю является безопасным (при этом ничего не происходит).

Создание динамического массива

Синтаксис выделения памяти для массива имеет вид:

указатель = new тип[размер];

В качестве размера массива может выступать любое целое положительное значение.

Пример.

```
#include <iostream>

int main()
{
std::cout << "Enter a positive integer: ";
    int length;
    std::cin >> length;
    int *array = new int[length]; /* используем оператор new[] для выделения массива.
Обратите внимание, переменная length не обязательно должна быть константой! */
    std::cout << "I just allocated an array of integers of length " << length << '\n';
array[0] = 7; // присваиваем элементу под номером 0 значение 7
    delete[] array; /* используем оператор delete[] для освобождения выделенной для
массива памяти; об этом я расскажу чуть позже*/
    array = 0; // используйте nullptr вместо 0 в C++
    return 0;
}
```

Создание динамического массива

Пример. Разработаем программу, в которой будет создаваться простая динамическая переменная.

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

int main(int argc, char* argv[])
{
 int *ptrvalue = new int; // динамическое выделение памяти под объект типа int
 *ptrvalue = 9; // инициализация объекта через указатель
 /*  int *ptrvalue = new int (9); инициализация может выполнятся сразу при объявлении динамического объекта */
 cout << "ptrvalue = " << *ptrvalue << endl;
 delete ptrvalue; // высвобождение памяти
 system("pause");
return 0;
}
```

В строке 10 показан способ объявления и инициализации значением "девять" динамического объекта: все, что нужно, так это указать значение в круглых скобочках после типа данных.

Результат работы программы:

ptrvalue = 9

Для продолжения нажмите любую клавишу …

Использование операции new для создания динамических массивов

Общая форма выделения и назначения памяти для массива выглядит следующим образом:

```
имя_типа *имя_указателя = new имя_типа [количество_элементов];
```

Вызов операции new выделяет достаточно большой блок памяти, чтобы в нем поместилось количество_элементов типа имя_типа, и устанавливает в имя_указателя указатель на **первый** элемент. Как вы вскоре увидите, **имя_указателя можно использовать точно так же, как обычное имя массива.**

Создание динамического массива

Пример. Разработаем программу, в которой будет создаваться простая динамическая переменная.

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

int main(int argc, char* argv[])
{
 int *ptrvalue = new int; // динамическое выделение памяти под объект типа int
 *ptrvalue = 9; // инициализация объекта через указатель
 /*  int *ptrvalue = new int (9); инициализация может выполнятся сразу при объявлении динамического объекта */
 cout << "ptrvalue = " << *ptrvalue << endl;
 delete ptrvalue; // высвобождение памяти
 system("pause");
return 0;
}
```

В строке 10 показан способ объявления и инициализации значением "девять" динамического объекта: все, что нужно, так это указать значение в круглых скобочках после типа данных.

Результат работы программы:

ptrvalue = 9

Для продолжения нажмите любую клавишу …

Создание динамического массива

```
#include <iostream>
using namespace std;

int main()
{
    int num; // размер массива
    cout << "Enter integer value: ";
    cin >> num; // получение от пользователя размера массива

    int *p_darr = new int[num]; // Выделение памяти для массива
    for (int i = 0; i < num; i++) {
        // Заполнение массива и вывод значений его элементов
        p_darr[i] = i;
        cout << "Value of " << i << " element is " << p_darr[i] << endl;
    }
    delete [] p_darr; // очистка памяти
    return 0;
}
```

Синтаксис выделения памяти для массива имеет вид

тип *указатель = new тип[размер]

Удаление динамического массива

При удалении динамических массивов используется специальная форма оператора delete для массивов, т.е. delete[]:

delete []; //указатель на динамический массив

Таким образом, мы сообщаем процессору, что ему нужно очистить память от нескольких переменных вместо одной.

Пример.

```
#include <iostream>

int main()
{
    int n = 5; // размер массива
    int *p = new int[n]{ 1, 2, 3, 4, 5 }; // массив состоит из чисел 1, 2, 3, 4
    for (int *q = p; q != p + n; q++)
    {
        std::cout << *q << "\t";
    }
std::cout << std::endl;
     delete [] p;
     return 0;
}
```

Создание и удаление динамических массивов

Создать динамический массив на C++ легко; вы сообщаете операции new тип элементов массива и требуемое количество элементов. Синтаксис, необходимый для этого, предусматривает указание имени типа с количеством элементов в квадратных скобках. Например, если необходим массив из 10 элементов int, следует записать так:

```
int * psome = new int [10] ; // получение блока памяти из 10 элементов типа int
```

Операция new возвращает адрес первого элемента в блоке. В данном примере это значение присваивается указателю psome.

Использование операции new для создания динамических массивов

Обратите внимание, что psome — это указатель на отдельное значение int, являющееся первым элементом блока. Отслеживать количество элементов в блоке возлагается на вас как разработчика. То есть, поскольку компилятор не знает о том,что psome указывает на первое из" 10 целочисленных значений, вы должны писать свою программу так, чтобы она самостоятельно отслеживала количество элементов.

На самом деле программе, конечно же, известен объем выделенной памяти, так что она может корректно освободить ее позднее, когда вы воспользуетесь операцией delete []. Однако эта информация не является открытой; вы, например, не можете применить операцию sizeof, чтобы узнать количество байт в выделенном блоке.

Общая форма выделения и назначения памяти для массива выглядит следующим образом:

```
имя_типа *имя_указателя = new имя_типа [количество_элементов] ;
```

Вызов операции new выделяет достаточно большой блок памяти, чтобы в нем поместилось количество_элементов типа имя_типа, и устанавливает в имя_ указателя указатель на первый элемент. Как вы вскоре увидите, имя_указателя можно использовать точно так же, как обычное имя массива.

Использование динамического массива

Как работать с динамическим массивом после его создания? Для начала подумаем о проблеме концептуально. Следующий оператор создает указатель psome, который указывает на первый элемент блока из 10 значений int:

```
int * psome = new int [10]; // получить блок для 10 элементов типа int
```

Представьте его как палец, указывающий на первый элемент. Предположим, что int занимает 4 байта. Перемещая палец на 4 байта в правильном направлении, вы можете указать на второй элемент. Всего имеется 10 элементов, что является допустимым диапазоном, в пределах которого можно передвигать палец. Таким образом, операция new снабжает всей необходимой информацией для идентификации каждого элемента в блоке. Теперь взглянем на проблему практически. Как можно получить доступ к этим элементам? С первым элементом проблем нет. Поскольку psome указывает на первый элемент массива, то * psome и есть значение первого элемента. Но остается еще девять элементов. Простейший способ доступа к этим элементам может стать сюрпризом для вас, если вы не работали с языком С; просто используйте указатель, как если бы он был именем массива. То есть можно писать psome [0] вместо *psome для первого элемента, psome [1] — для второго и т.д. Получается, что применять указатель для доступа к динамическому массиву очень просто, хотя не вполне понятно, почему этот метод работает. Причина в том, что С и С++ внутренне все равно работают с массивами через указатели.

Как всегда, вы должны сбалансировать каждый вызов new соответствующим вызовом delete, когда программа завершает работу с этим блоком памяти. Однако использование new с квадратными скобками для создания массива требует применения альтернативной формы delete при освобождении массива:

delete [] psome; // *освобождение динамического массива*

Присутствие квадратных скобок сообщает операционной системе, что она должна освободить весь массив, а не только один элемент, на который указывает указатель. Обратите внимание, что скобки расположены между delete и указателем.

Использование динамического массива

Подобная эквивалентность указателей и массивов — одно из замечательных свойств C и C++. (Иногда это также и проблема, но это уже другая история.) Ниже об этом речь пойдет более подробно. В листинге ниже показано, как использовать new для создания динамического массива и доступа к его элементам с применением нотации обычного массива. В нем также отмечается фундаментальное отличие между указателем и реальным именем массива.

```
#include <iostream>
using namespace std;
int main()
{
double * p3 = new double [3]; // пространство для 3 значений double
p3[0] = 0.2; // трактовать p3 как имя массива
p3[1] = 0.5 ;
p3[2] = 0.8 ;
cout <<"p3[1] is " << p3[1] <<".\n"; // вывод p3[1]
p3 = p3 + 1;  // увеличение указателя
cout << "Nowp3[0] is " <<p3[0] << " and  "; // выводp3[0]
cout << "p3[1] is " <<p3[1] << " .\n" ;  // выводp3[1]
p3 = p3 - 1;  // возврат указателя в начало
delete [] p3;  // освобождение памяти
return 0;
}
```

```
p3[1] is 0.5.
Nowp3[0] is 0.5 and  p3[1] is 0.8 .
```

Использование динамического массива

Как видите, здесь использует указатель p3, как если бы он был именем массива: p3 [0] для первого элемента и т.д. Фундаментальное отличие между именем массива и указателем проявляется в следующей строке:

p3 = p3 + 1; // допускается для указателей, но не для имен массивов

Вы не можете изменить значение для имени массива. Но указатель — переменная, а потому ее значение можно изменить. Обратите внимание на эффект от добавления 1 к p3. Теперь выражение p3 [0] ссылается на бывший второй элемент массива. То есть добавление 1 к p3 заставляет p3 указывать на второй элемент вместо первого. Вычитание 1 из значения указателя возвращает его назад, в исходное значение, поэтому программа может применить delete[] с корректным адресом.

Действительные адреса соседних элементов int отличаются на 2 или 4 байта, поэтому тот факт, что добавление 1 к p3 дает адрес следующего элемента, говорит о том, что арифметика указателей устроена специальным образом. Так оно и есть на самом деле.

Использование динамического массива

Тут то же самое, но чуть более по-русски..

```
// arraynew.cpp -- использование операции new для массивов
#include <iostream>
int main()
{
   using namespace std;
   double * p3 = new double [3];   // пространство для 3 double
   p3[0] = 0.2;                    // трактовать p3 как имя массива
   p3[1] = 0.5;
   p3[2] = 0.8;
   cout << "p3[1] равно " << p3[1] << ".\n";
   p3 = p3 + 1;       // увеличить указатель
   cout << "Теперь p3[0] равно " << p3[0] << " и ";
   cout << "p3[1] равно " << p3[1] << ".\n";
   p3 = p3 - 1;       // вернуть указатель к началу
   delete [] p3;      // освободить память
   return 0;
}
```

Вывод этой программы выглядит следующим образом:

```
p3[1] равно 0.5.
Теперь p3[0] равно 0.5 и p3[1] равно 0.8.
```

Как видите, `arraynew.cpp` использует указатель `p3`, как если бы он был именем массива, с первым элементом `p3[0]` и так далее. Фундаментальное отличие между именем массива и указателем проявляется в следующей строке:

```
p3 = p3 + 1;       // допускается для указателей, но не для имен массивов
```

Вы не можете изменить значение для имени массива. Но указатель — переменная, а потому ее значение можно изменить. Обратите внимание на эффект от добавления 1 к p3. Теперь выражение p3[0] ссылается на бывший второй элемент массива. То есть добавление 1 к p3 заставляет p3 указывать на второй элемент вместо первого. Вычитание 1 из значения указателя возвращает его назад, в исходное значение, поэтому программа может применить delete[] с корректным адресом.

Этот пример показывает изменение способа доступа к информации.

Арифметика указателей

Адресная арифметика (address arithmetic) - это способ вычисления адреса какого-либо объекта при помощи арифметических операций над указателями, а также использование указателей в операциях сравнения. Адресную арифметику также называют арифметикой над указателями (pointer arithmetic).

Арифметика указателей

Согласно стандартам языка Си и Си++, при арифметике с указателями, результирующий адрес должен оставаться строго на границе единичного объекта массива (или следовать сразу за ним). Сложение или вычитание указателя сдвигает его на величину, кратную размеру того типа данных, на который он указывает.

Пример. Пусть есть указатель на массив 4-байтных целых. Инкремент этого указателя приведет к увеличению его значения на 4 (размер элемента). Такой эффект часто используется для увеличения указателя для того, чтобы он указывал на следующий элемент в смежном массиве целых чисел.

Арифметика указателей

Арифметика с указателями не может быть применена к указателям на неизвестные типы, поскольку неизвестные типы не имеют размера, и соответственно адрес, на который ссылается указатель, не может быть прибавлен к нему. Однако, иногда существуют нестандартные расширения компилятора, позволяющие выполнять байтовую арифметику на нетипизированных указателях (void *).

Арифметика указателей

Указатели и целочисленные переменные не являются взаимозаменяемыми объектами. Константа нуль — единственное исключение из этого правила: ее можно присвоить указателю, и указатель можно сравнить с нулевой константой. Чтобы показать, что нуль — это специальное значение для указателя, вместо цифры нуль, как правило, записывают константу NULL.

Арифметика указателей

Арифметика с указателями дает программисту возможность работать с разными типами одинаковым способом: за счет сложения и вычитания необходимого числа элементов вместо действительного сдвига байтов. В частности, описание языка Си явно задает равнозначность синтаксической структуры A[i], которая является i-ым элементом массива A, и *(A+i), которая представляет собой содержание элемента, на который указывает выражение A+i. Также подразумевается, что i[A] равнозначно A[i].

Действительные адреса соседних элементов int отличаются на 2 или 4 байта, поэтому тот факт, что добавление 1 к p3 дает адрес следующего элемента, говорит о том, что арифметика указателей устроена специальным образом. Так оно и есть на самом деле.

a[i] = *(a+i) = i[a] - интересное следствие из арифметики указателей

Арифметика указателей

Из-за сложностей использования указателей многие современные языки программирования высокого уровня (например, Java или C#) не разрешают прямой доступ к памяти с использованием адресов.

Удаление динамического массива

При удалении динамических массивов используется специальная форма оператора delete для массивов, т.е. delete[]:

delete []; //указатель на динамический массив

Таким образом, мы сообщаем процессору, что ему нужно очистить память от нескольких переменных вместо одной.

Пример.

```
#include <iostream>

int main()
{
    int n = 5; // размер массива
    int *p = new int[n]{ 1, 2, 3, 4, 5 }; // массив состоит из чисел 1, 2, 3, 4, 5
    for (int *q = p; q != p + n; q++)
    {
        std::cout << *q << "\t";
    }
std::cout << std::endl;
     delete [] p;
     return 0;
}
```

Инициализация динамического массива

Если вы хотите инициализировать динамический массив значением 0, то всё довольно просто:

int *array = new int[length]();

Есть возможность инициализации динамических массивов через списки инициализаторов:

int fixedArray[5] = { 9, 7, 5, 3, 1 }; // инициализируем фиксированный массив
int *array = new int[5] { 9, 7, 5, 3, 1 }; // инициализируем динамический массив

После создания динамического массива мы сможем с ним работать по полученному указателю, получать и изменять его элементы.

Пример.

```
int n = 5; // размер массива
int *p = new int[n]{ 1, 2, 3, 4, 5 };
for (int *q = p; q != p + n; q++)
{
    std::cout << *q << "\t";
}
```

Пример. Разработаем программу, в которой создадим одномерный динамический массив, заполненный случайными числами.

```
#include "stdafx.h"
#include <iostream>
// в заголовочном файле <ctime> содержится прототип функции time()
#include <ctime>
// в заголовочном файле <iomanip> содержится прототип функции setprecision()
#include <iomanip>
using namespace std;

int main(int argc, char* argv[])
{
   srand(time(0)); // генерация случайных чисел
   float *ptrarray = new float [10]; /* создание динамического массива вещественных чисел на десять элементов*/
      for (int count = 0; count < 10; count++)
         ptrarray[count] = (rand() % 10 + 1) / float((rand() % 10 + 1)); /*заполнение массива случайными числами с масштабированием от 1 до 10 */
      cout << "array = ";
      for (int count = 0; count < 10; count++)
         cout << setprecision(2) << ptrarray[count] << "    ";
      delete [] ptrarray; // высвобождение памяти
      cout << endl;
   system("pause");
   return 0;
}
```

Созданный одномерный динамический массив заполняется случайными вещественными числами, полученными с помощью функций генерации случайных чисел, причём числа генерируются в интервале от 1 до 10, интервал задается так — rand() % 10 + 1. Чтобы получить случайные вещественные числа, выполняется операция деления, с использованием явного приведения к вещественному типу знаменателя — float((rand() % 10 + 1)). Чтобы показать только два знака после запятой используем функцию setprecision(2), прототип данной функции находится в заголовочном файле <iomanip>. Функция time(0) присваивает параметрам генератора случайных чисел значения, зависящие от системного времени, что позволяет воспроизводить случайность возникновения чисел. Вот что, например, может появиться на экране в результате выполнения программы:

array = 0.8 0.25 0.86 0.5 2.2 10 1.2 0.33 0.89 3.5
Для продолжения нажмите любую клавишу . . .

По завершению работы с массивом, он удаляется, таким образом, высвобождается память, отводимая под его хранение.

Пример.

```
// объявление двумерного динамического массива на 10 элементов:
float **ptrarray = new float* [2]; // две строки в массиве
    for (int count = 0; count < 2; count++)
        ptrarray[count] = new float [5]; // и пять столбцов
/* где ptrarray – массив указателей на выделенный участок памяти под массив вещественных чисел типа float*/
```

Сначала объявляется указатель второго порядка float **ptrarray, который ссылается на массив указателей float* [2], где размер массива равен двум. После чего в цикле for каждой строке массива, объявленного в строке 2 выделяется память под пять элементов. В результате получается двумерный динамический массив ptrarray[2][5].

Пример. Рассмотрим пример высвобождения памяти, отводимой под двумерный динамический массив:

```
// высвобождение памяти, отводимой под двумерный динамический массив:
    for (int count = 0; count < 2; count++)
        delete [] ptrarray[count];
//      где 2 – количество строк в массиве
```

Объявление и удаление двумерного динамического массива выполняется с помощью цикла, как показано ранее.

Работа с динамическими массивами. Двумерный массив

Пример. Разработаем программу, в которой создадим двумерный динамический массив.

```
// new_delete_array2.cpp: определяет точку входа для консольного приложения.
#include "stdafx.h"
#include <iostream>
#include <ctime>
#include <iomanip>
using namespace std;

int main(int argc, char* argv[])
{
   srand(time(0)); // генерация случайных чисел
   // динамическое создание двумерного массива вещественных чисел на десять элементов
   float **ptrarray = new float* [2]; // две строки в массиве
   for (int count = 0; count < 2; count++)
      ptrarray[count] = new float [5]; // и пять столбцов
   // заполнение массива
   for (int count_row = 0; count_row < 2; count_row++)
      for (int count_column = 0; count_column < 5; count_column++)
         ptrarray[count_row][count_column] = (rand() % 10 + 1) / float((rand() % 10 + 1)); //заполнение массива случайными
числами с масштабированием от 1 до 10
   // вывод массива
   for (int count_row = 0; count_row < 2; count_row++)
   {
      for (int count_column = 0; count_column < 5; count_column++)
         cout << setw(4) <<setprecision(2) << ptrarray[count_row][count_column] << "   ";
      cout << endl;
   }
   // удаление двумерного динамического массива
   for (int count = 0; count < 2; count++)
      delete []ptrarray[count];
   system("pause");
   return 0;
}
```

При выводе массива была использована функция setw(), которая отводит место заданного размера под выводимые данные. В нашем случае, под каждый элемент массива по четыре позиции, это позволяет выровнять, по столбцам, числа разной длины:

```
2.7    10  0.33    3   1.4
  6  0.67  0.86   1.2  0.44
Для продолжения нажмите любую клавишу . . .
```

Трехмерные массивы:

```
int x = 5;
int y = 5;
int z = 4;
/* размеры массива */
int ***arr = new int**[x];
for(int i=0;i<x;i++)
        {
        arr[i]=new int*[y];
        for(int j=0;j<y;j++) arr[i][j]=new int[z];
        }
```

Удаление трехмерных массивов производиться так:

```
for(int i=0;i<x;i++)
        {
        for(int j=0;j<y;j++)
                {
                delete[]arr[i][j];
                }
        delete[]arr[i];
        }
delete[]arr;
```

Четырехмерный массив:

```
int x=3,y=3,z=3,t=3;
int ****arr = new int***[x];
for(int i=0;i<x;i++)
        {
        arr[i] = new int**[y];
        for(int j=0;j<y;j++)
                {
                arr[i][j] = new int*[z];
                for(int k=0;k<z;k++)
                        {
                        arr[i][j][k] = new int[t];
                        }
                }
        }
```

Мы создавали только массивы типа int. Также можно создавать и массивы других типов. Ниже приводится динамический массив типа float:

```
int x = 3;
int y = 7;
float **arr = new float*[x];
for(int i=0;i<x;i++) arr[i]=new float[y];
```

Удаляется он точно так же.

Правила использования операций new и delite для создания динамических массивов

• Не использовать delete для освобождения той памяти, которая не была выделена new.

• Не использовать delete для освобождения одного и того же блока памяти дважды.

• Использовать delete[], если применялась операция new[] для размещения массива.

• Использовать delete без скобок, если применялась операция new для размещения отдельного элемента.

• Помнить о том, что применение delete к нулевому указателю является безопасным (при этом ничего не происходит).

ВИДЫ МАССИВОВ В РАЗНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Существует четыре вида массивов:
1) динамические массивы
2) массивы переменной длины
3) массив фиксированной длины
4) статический массив

Рассмотрим четыре массива более подробно.
1) **Динамический массив**
Для него должен существовать метод, который непосредственно меняет длину

2) **Массив переменной длины**
Массив переменной длины является частным случаем динамического массива. Для него тоже существует метод, который непосредственно меняет длину, но он более сложный

3) **Массив фиксированной длины**
Массив фиксированной длины является частным случаем массива переменной длины. В этом случае после присвоения значения длины мы больше не можем ее изменять.

4) **Статический массив**
Статический массив является частным случаем массива фиксированной длины.

ДИНАМИЧЕСКИЙ МАССИВ И МАССИВ ПЕРМЕННОЙ ДЛИНЫ

В Википедии есть два отдельных листа об этом:

1) относительно динамических массивов: https://en.wikipedia.org/wiki/Dynamic_array

2) относительно массивов переменной длины: https://en.wikipedia.org/wiki/Variable-length_array

Массивы переменной длины имеют переменные **размеры, которые устанавливаются один раз** во время выполнения.

Динамические массивы также являются массивами переменной длины, но они **также могут изменять размер (re-dimension) после их создания** . Это позволяет массиву расти, чтобы вместить дополнительные элементы выше его первоначальной емкости. При использовании массива вам придется вручную изменить размер массива или перезаписать существующие данные.

Массив переменной длины

Динамическим называется массив, размер которого может изменяться во время исполнения программы. Возможность изменения размера отличает динамический массив от статического, размер которого задаётся на момент компиляции программы. Для изменения размера динамического массива язык программирования, поддерживающий такие массивы, должен предоставлять встроенную функцию или оператор. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объёмы данных, а регулировать размер массива в соответствии с реально необходимыми объёмами.

Массив переменной длины

Также иногда к динамическим относят *массивы переменной длины*, размер которых не фиксируется при компиляции, а задаётся при создании или инициализации массива во время исполнения программы. От «настоящих» динамических массивов они отличаются тем, что для них не предоставляются средства автоматического изменения размера с сохранением содержимого, так что при необходимости программист должен реализовать такие средства самостоятельно.

В самом языке Си нет динамических массивов, но функции стандартной библиотеки malloc, free и realloc позволяют реализовать массив переменного размера:

```
int *mas = (int*)malloc(sizeof(int) * n); // Создание массива из n элементов типа int
... mas = (int*)realloc(mas, sizeof(int) * m); // Изменение размера массива с n на m с сохранением содержимого
... free(mas); // Освобождение памяти после использования массива
```

Неудобство данного подхода состоит в необходимости вычислять размеры выделяемой памяти, применять явное преобразование типа и тщательно отслеживать время жизни массива (как и всегда при работе с динамически выделенной памятью в Си).

Многомерный динамический массив может быть создан как массив указателей на массивы:

```
int **A = (int **)malloc(N*sizeof(int *));
for(int i = 0; i < N; i++)
{
A[i] = (int *)malloc(M*sizeof(int));
}
```

Однако рост размерности существенно усложняет процедуры создания массива и освобождения памяти по завершении его использования. Ещё более усложняется задача изменения размера массива по одной или нескольким координатам.

Массив переменной длины

В C++ поддерживаются функции работы с памятью из стандартной библиотеки Си, но их использование не рекомендуется. Массив переменной длины здесь также можно выделить с помощью стандартных команд работы с динамической памятью new и delete:

```
// Создание массива длиной n
int *mas = new int[n];
...
// Освобождение памяти массива delete []mas;
```

Как и в случае с Си, здесь требуется отслеживать время жизни массива, чтобы избежать утечки памяти или, наоборот, преждевременного освобождения памяти. Аналога realloc здесь нет, так что изменить размер массива можно только вручную, выделив новую память нужного размера и перенеся в неё данные.

Библиотечным решением является шаблонный класс std::vector:

Приведем *пример* массива переменной длины:

```
void f(int dim)
 {
char str[dim]; /* символьный массив переменной длины */
/* ... */
}
```

Здесь размер массива str определяется значением переменной dim, которая передается в функцию f() как параметр. Таким образом, при каждом вызове f() создается массив str разной длины.

Тест на тему «Динамические массивы и переменные». Проверьте себя!

https://codelessons.ru/cplusplus/dinamicheskie-massivy-i-peremennye-vse-samoe-glavnoe.html

Литература

1. «Уроки C++ с нуля» https://code-live.ru
2. «Язык программирования C++» http://cppstudio.com/post/432/
3. Процедурное программирование на языках СИ и C++ : учебно-методическое пособие / Л. А. Скворцова [и др.]. — М.: РТУ МИРЭА, 2018. — 238 с. [Электронный ресурс]. Режим доступа: https://library.mirea.ru/books/53585
4. Трофимов В.В., Павловская Т.А. Алгоритмизация и программирование: учебник для академического бакалавриата. М.: Издательство Юрайт, 2017
5. [Электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/16740/1301/info
6. Уроки C++ с нуля. [Электронный ресурс]. Режим доступа: https://code-live.ru/tag/cpp-manual,
7. Введение в языки программирования Си C++. [Электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/1039/231/info