МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра информационных систем

ОТЧЕТ

по практической работе №3

по дисциплине «Программирование»

Тема: Указатели и многомерные статические массивы

Студентка гр. 0324 Серебрякова А.К.

Преподаватель Глущенко А.Г.

Санкт-Петербург

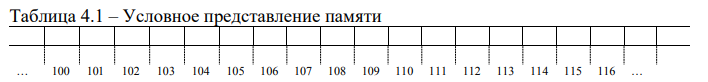
2020

Цель работы.

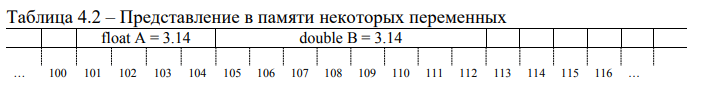
Изучение структуры многомерных статических массивов, обработка данных многомерных массивов. Получение практических навыков работы с указателями. Изучение простейшей арифметики указателей.

Основные теоретические положения.

Понятие указателя Компилятор, обрабатывая оператор определения переменной, выделяет память в соответствии с типом переменной и инициализирует её указанным значением. Все обращения по имени переменной заменяются компилятором на адрес области памяти, в которой хранится значение переменной. Возможно создание собственных переменных, которые будут хранить какой-то адрес памяти. Такие переменные называются указателями.

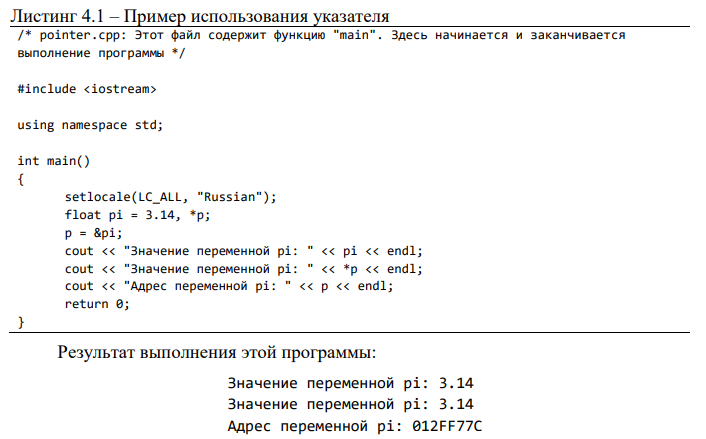


Указатели предназначены для хранения адресов областей памяти (табл. 4.1.). В C++ существует три вида указателей: 1) Указатели на объект, который содержит адрес области памяти, хранящей данные определенного типа. 2) Указатели на функцию. Указатель на функцию содержит адрес сегмента кода, по которому располагается исполняемый код функции. Указатели на функции используются для косвенного вызова функции (через обращение к переменной, хранящей её адрес), а также для передачи имени функции в другую функцию в качестве параметра. Указатель функции должен иметь тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа. 3) Указатели на void. Такой указатель применяется в тех случаях, когда тип объекта, адрес которого нужно хранить, не определен. Указателю типа void 46 можно присвоить значение любого типа, но перед выполнением каких-либо действий его нужно явным образом преобразовать к этому типу. Для получения адреса какого-либо программного объекта используют оператор &. Предположим, что у нас имеется две переменные: float A = 3.14 и double = 3.14 (табл. 4.2). Видно, что обе переменные занимают по 4 байта памяти. Они имеют адреса 101 и 105 соответственно.



Если использовать оператор &, то он вернет адреса этих переменных. Правда, обычно значение адреса памяти выводится в шестнадцатеричном коде. Указатели на объекты определяются в программе следующим образом: \*; Тип данных – базовый тип указателя. Имя переменной – идентификатор переменной-указателя. Признаком того, что переменная – указатель, является символ ‘∗’, располагающийся перед именем переменной. Определим две переменных-указателя: float \*p1; int \*p2; Указатель 𝑝1 предназначен для хранения адресов участков памяти, размер которого соответствует типу int, а переменная 𝑝2 - для хранения адресов участков памяти, размер которых соответствует типу double. Указатели представляют собой обычные целые числа значения типа int и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

Инициализация указателей.

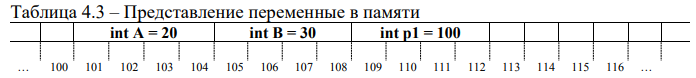


Присвоить указателю адрес некоторой переменной можно инструкцией присваивания и операции &. Пример инициализации указателя представлен в 47 листинге 4.1. Получить значение объекта, на который ссылается некоторый указатель можно с помощью операции \* (разыменовывание указателя).

Из памяти по адресу, который хранится в указатели, берется столько байт памяти, сколько требуется базовому типу указателя. Далее с этими байтами работают, как со значением базового типа указателя. С помощью указателей можно не только получать значения, расположенные по адресам, хранящимся в указателях, но и записывать нужные значения по этим адресам: \*p = 1.618; Указатели могут использоваться в различных выражениях наравне с обычными переменными и константами. При использовании указателей в выражениях важно помнить, что операция \* имеет наивысший приоритет по отношению к другим операциям (за исключением операции унарный минус). Указателю можно присвоить значение другого указателю, если совпадают их базовые типы. Хотя указатели представляют собой целые значения, присваивать им произвольные целые значения нельзя. Единственным исключением является присвоение указателю нулевого значения. Нулевое значение указателя означает то, что указатель не на что не ссылается.

Арифметика указателей

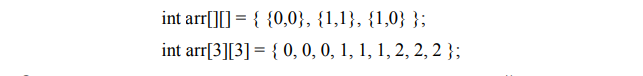
К указателям можно применять некоторые арифметические операции, одни из них +, −, ++, − −. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.



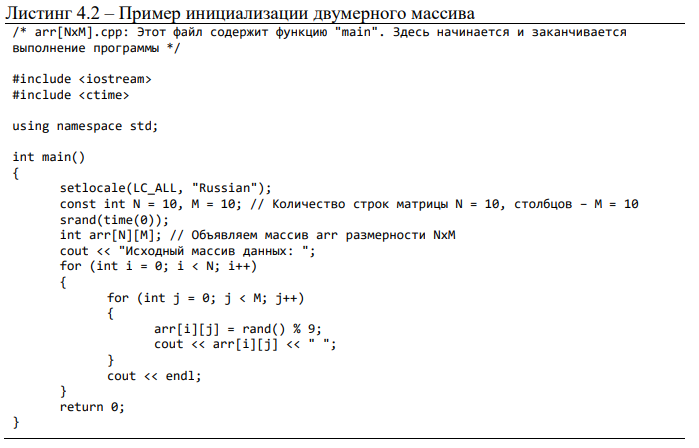
Указатель p1 содержит адрес переменной A. Если выполнить операцию p1 = p1 + 1; (или p1++;) Значение указателя изменится, и станет равно 104. Таким образом, добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. При этом добавлять или вычитать из указателей можно только целые значения. Не стоит забывать о приоритете операций: \*p1 + 1; и \*(p1 + 1); имеют совершенно разный смысл.

Указатели и массивы

Между массивами и указателями имеется очень тесная связь. Когда в программе определяется некоторый массив. Имя переменной без индексов представляет собой указатель на первый элемент массива: int Arr[5]; 49 Если вывести на экран значение переменной Arr, то будет отображено некоторое целое значение в шестнадцатеричном формате, которое соответствует адресу первого элемента этого массива. Именно из-за этого операция присвоения сразу всех значений одного массива в другому запрещена в C++. Попытка присвоения Arr1 = Arr2; привела бы к тому, что переменная Arr1, стала бы указывать на ту же область памяти, что и переменная Arr2. Адрес, который ранее хранился в переменной Arr1, был бы утерян, что привело бы к утечке памяти. Более того, по этой причине запрещены любые изменения значения переменной массива. Указателю, который имеет такой же базовый тип, как и элемент массива, можно присвоить указатель на массив. Но обратное присвоение выполнить невозможно, так как переменная массива – это константа, изменение которой запрещено. Так как переменная массива является указателем на первый элемент массива, появляются дополнительные возможности по работе с массивами на основе использования арифметики указателей. Например, Arr[4] эквивалентно \*(Arr + 4). Первое выражение – это пример обычной индексации элементов массива. Во втором выражении использовалась арифметика указателей, и с помощи операции + был получен адрес пятого элемента массива. Здесь нельзя забывать о приоритете операций. Использование арифметики указателей при работе с массивами приводит обычно к уменьшению объема генерируемого кода программы и к уменьшению времени ее выполнения, то есть к увеличению быстродействия. Поскольку указатель и имя массива взаимосвязаны, указатели можно индексировать, как обычные массивы, а также создавать массивы указателей.

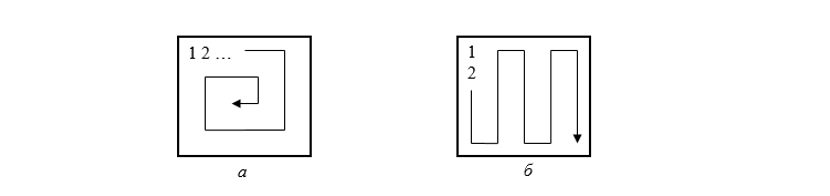
Объявление многомерных массивов Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому измерению указывается отдельно в квадратичных 50 скобках. Элементы многомерных массивов располагаются друг за другом в непрерывном участке памяти. При инициализации многомерного массива он представляется либо как массив из массивов, при этом каждый массив заключается в свои фигурные скобки (в этом случае левую размерность при описании можно не указывать), либо задается общий список элементов в том порядке, в котором элементы располагаются в памяти: 

Определить, сколько памяти выделено под многомерный массив, можно аналогично одномерному массиву с помощью операции sizeof. Необходимо определить, сколько выделяется памяти на 1 элемент массива и умножить на количество элементов массива. В листинге 4.2 представлен пример заполнения многомерного массива.

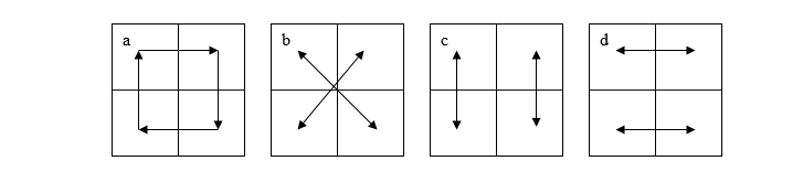


В приведенном выше примере сначала объявляется двумерный массив, затем при помощи двух циклов (один из которых вложенный) происходит заполнение массива случайными числами. Первый цикл нужен, чтобы пройти все строчки 51 матрицы, а второй цикл – пройти все строчки, заполнить их значениями и вывести на экран.

Постановка задачи.

1)    Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до  N\*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы. 

2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

Выполнение работы.

1) Для выполнения этого задания я использовала High-Level Input and Output Functions

2) Чтобы поменять блоки местами, создала 4 массива n/2 \* n/2 для хранения всех блоков массива n\*n. С помощью цикла for присваивала нужной части большого массива значения маленького массива.

3) Сортировка массива bubble sort при помощи указателей.

4) Через цикл for, при помощи арифметики указателей меняла (действие зависит от выбора пользователя) каждый элемент массива на введенное число.

Вывод

Была создана программа , решающая задания для двумерного массива. Узнала о работе с массивом при помощи арифметики указателей.

Примеры работы

1) Вывод спиралью

2) Вывод змейкой

3 – 6) Переставление блоков

7) Сортировка

8 – 11) Изменение массива на заданное число

