**Глава 9. Массивы. Методы сортировки массивов**

**9.1 Операции с массивами**

Массив — структура данных, хранящая набор значений, идентифицируемых по индексу. Особенностью массива является константная вычислительная сложность доступа к элементу массива по индексу. Размерность массива — это количество индексов, необходимое для однозначной адресации элемента. Массивы бывают одномерные, двумерные, трёхмерные и так далее.

В алгоритмическом языке программирования C++ индексация элементов осуществляется целыми числами, идущими по порядку, начиная с нуля. Таким образом, первый элемент массива имеет индекс, равный нулю. Например, имеем массив «m» на десять элементов с уже имеющимися произвольными значениями (рисунок 9.1). При обращении к элементу массива необходимо указать его индекс, который может иметь вид константы, переменной или выражения. Массив относится к структурам данных с произвольным доступом. Это значит, что к любому элементу последовательности можно обратиться за равные промежутки времени, не зависящие от размеров последовательности.



Рисунок 9.1 – Способы обращения к элементам массива

При необходимости, элементы массива можно задать при его определении. Если количество объявленных элементов будет меньше размера массива, то оставшимся элементам будет присвоено нулевое значение. Если не указывать размерность массива, память будет зарезервирована по количеству заданных элементов (рисунок 9.2).



Рисунок 9.2 – Способы объявление размерности массива

На языке программирования C++ массив можно сформировать с произвольными значениями с помощью функции rand(), которая возвращает псевдослучайное значение из промежутка от 0 до 32767 (рисунок 9.3). С использованием дополнительных операторов можно корректировать промежуток возвращаемых значений. Для использования этой функции необходимо подключить библиотечный файл <stdlib.h>.

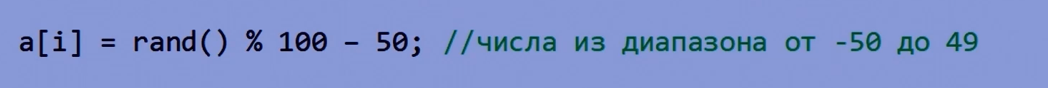


Рисунок 9.3 – Пример фрагмента кода для случайного объявления элементов массива

Для нахождения максимального элемента массива можно воспользоваться методом линейного поиска.

Например: объявляется массив из 10 элементов, далее каждый элемент массива сравнивается со всеми остальными. Если текущий элемент больше максимального – он становится максимальным, если нет, то проверяются следующие элементы массива до его конца (рисунок 9.4).

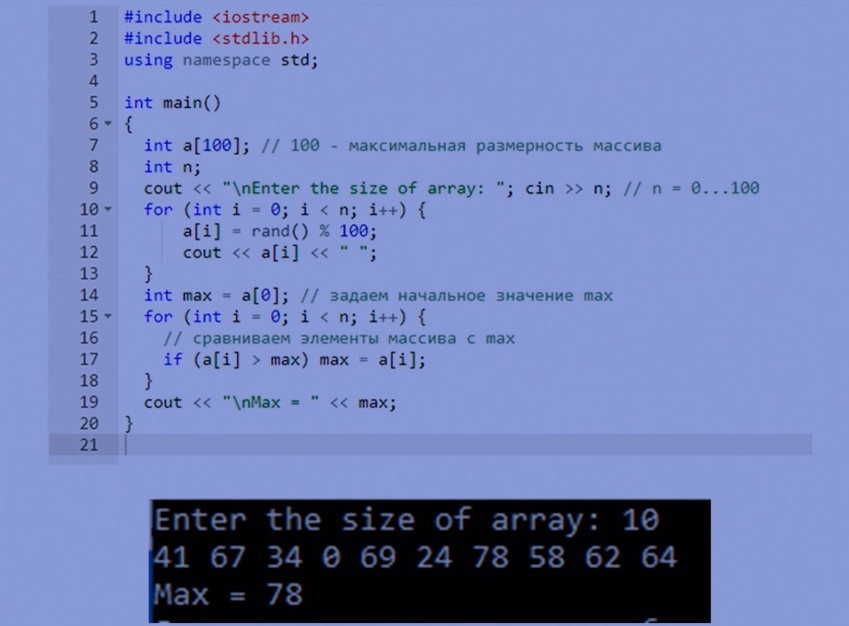


Рисунок 9.4 – Код программы для нахождения максимального элемента

Чтобы найти сумму элементов массива с четными индексами, необходимо проверять индексы элементов массива и в сумму добавлять только элементы, которые имеют четные индексы. Второй способ выполняет сложение элементов массива через один, причём первым элементом, который должен быть в сумме, будет элемент с индексом, равным нулю (рисунок 9.5).

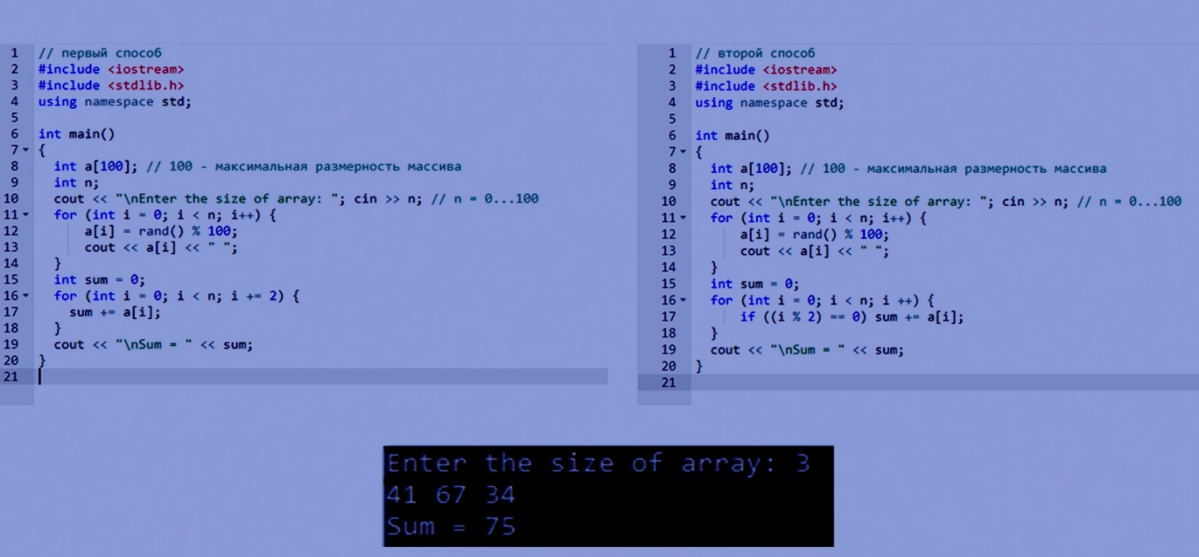


Рисунок 9.5 – Фрагмент кода программы для нахождения суммы элементов массива с четными индексами

Для объявления многомерного массива требуется указать размерность каждого измерения в квадратных скобках. В качестве примера двумерного массива можно привести математическую матрицу, которая содержит значения элементов на пересечении строк и столбцов. Чтобы получить доступ к элементу многомерного массива, необходимо указать все его индексы –индекс строки и столбца (рисунок 9.6).

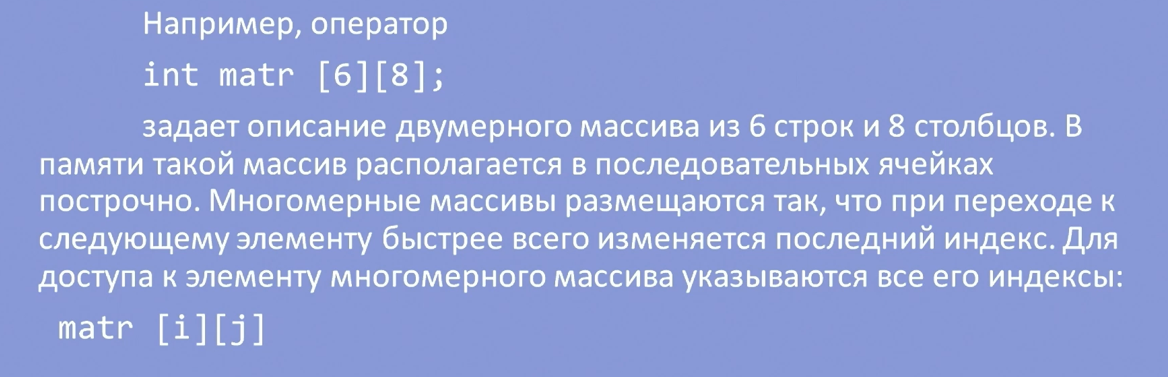


Рисунок 9.6 – Пример объявления двумерного массива

Многомерный массив на языке C++ представляется как массив массивов или как общий список элементов в порядке расположения в памяти. Многомерный массив также заполняется значениями, как и одномерный массив, изменяется только количество циклических процессов. Например, для заполнения двумерного массива потребуется два циклических процесса: по перебору строк и столбцов матрицы (рисунок 9.7).

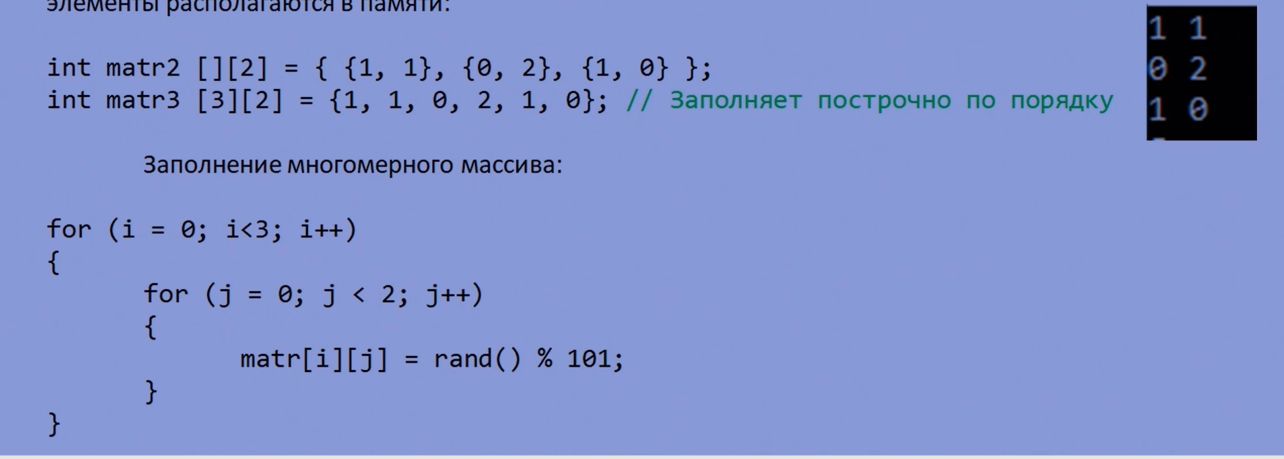


Рисунок 9.7 – Фрагмент кода программы для заполнения многомерного массива

Таблицу умножения можно представить, как сочетание строк и столбцов. Для ее вывода на экран через массив, необходимо сначала заполнить его нужными значениями, связав индексы с цифрами таблицы умножения, а затем вывести в правильном порядке с помощью операторов цикла (рисунок 9.8).

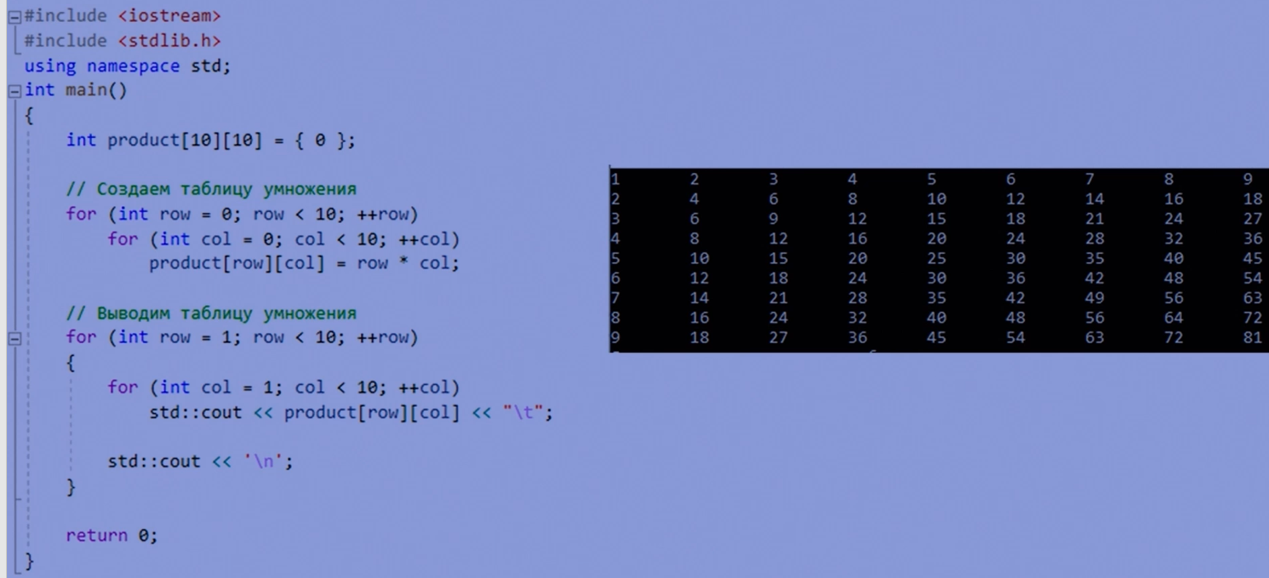


Рисунок 9.8 – Фрагмент кода для вывода таблицы умножения на экран

Для нахождения суммы элементов строк двумерного массива можно создать дополнительный массив для записи в него сумм строк, с помощью операторов цикла поочередно найти сумму каждой строки и вписать ее в дополнительный массив, а затем вывести на экран (рисунок 9.9).



Рисунок 9.9 – Фрагмент кода для вывода суммы элементов строк двумерного массива на экран

Для закрепления материала представлены четыре практические задачи:

1. Задан одномерный массив. За один просмотр найти значения максимального и минимального элементов массива.
2. Задан одномерный массив. Проверить, упорядочен ли он по возрастанию значений элементов.
3. Задан одномерный массив. Перевернуть этот массив таким образом, чтобы первый его элемент стал последним, второй- предпоследним и так далее.
4. Задан двумерный массив. Найти значение максимального элемента массива, а также индексы его строки и столбца.

Ответы к практическим задачам с пояснениями:

Первая задача: чтобы найти максимальный и минимальный элементы массива за один просмотр, необходимо поставить два условия для проверки текущего элемента. Если он меньше минимального или больше максимального, то соответственно данный элемент занимает его место. Если нет, то элементы массива перебираются дальше до конца массива (рисунок 9.10).

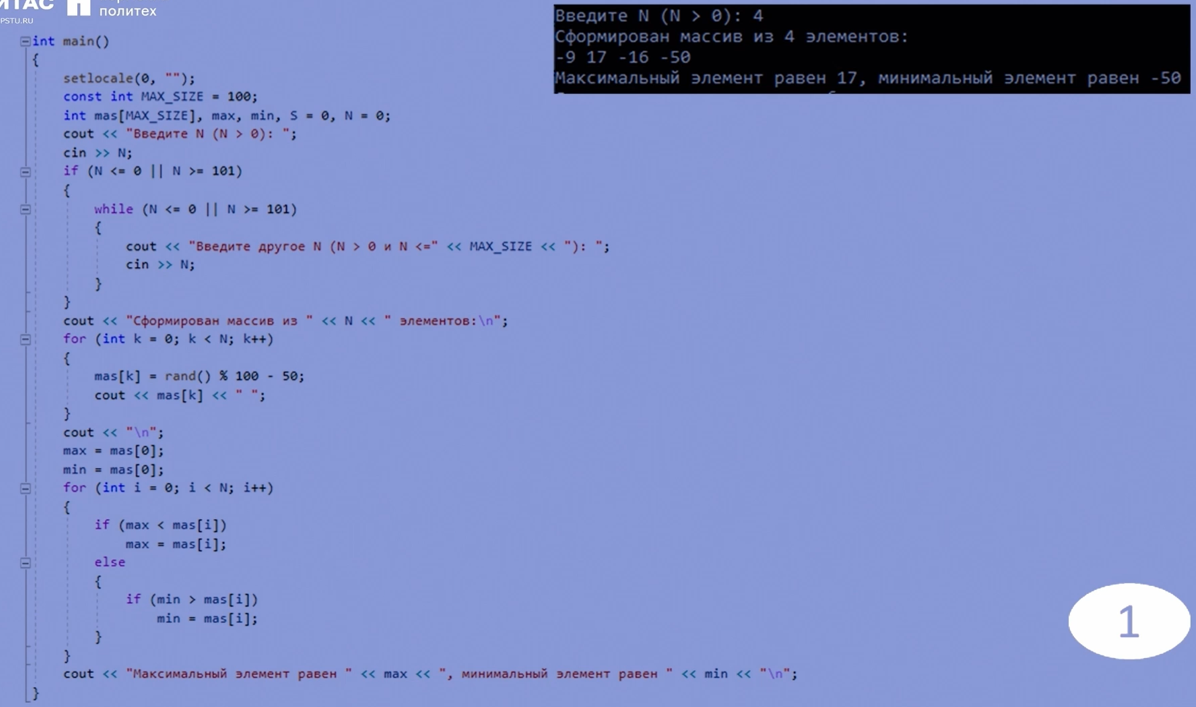


Рисунок 9.10 - Фрагмент кода к заданию 1

Вторая задача: чтобы проверить упорядочен ли массив по возрастанию, нужно запустить цикл проверки текущего элемента и следующего и добавить флаг. Если следующий элемент больше текущего, то проверяются следующие пары элементов. Если нет, то флаг меняет свое значение на false и в соответствии со значением флага выводится ответ – если значение флага true, то массив упорядочен, если false, значит массив является неупорядоченным (рисунок 9.11).

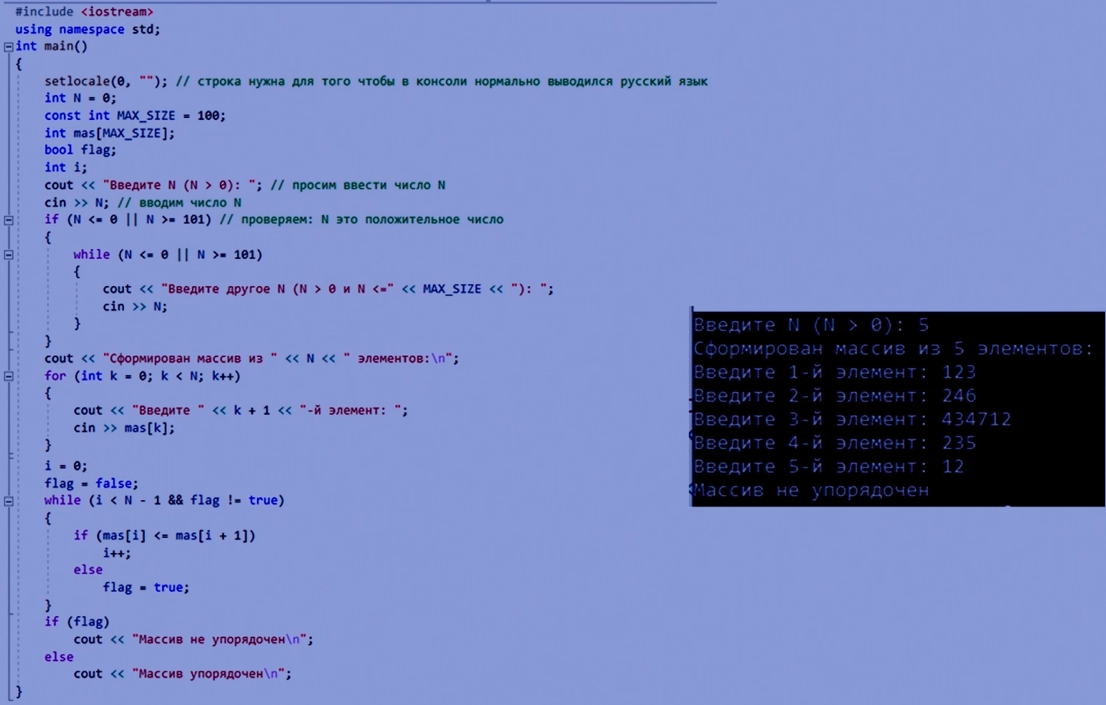


Рисунок 9.11 – Фрагмент кода к заданию 2

Третья задача: чтобы перевернуть одномерный массив, необходимо запустить цикл до середины массива и с помощью дополнительной переменной менять крайние нетронутые значения в массиве местами, пока индекс меняющихся элементов не будет больше или равен индексу середины (рисунок 9.12).

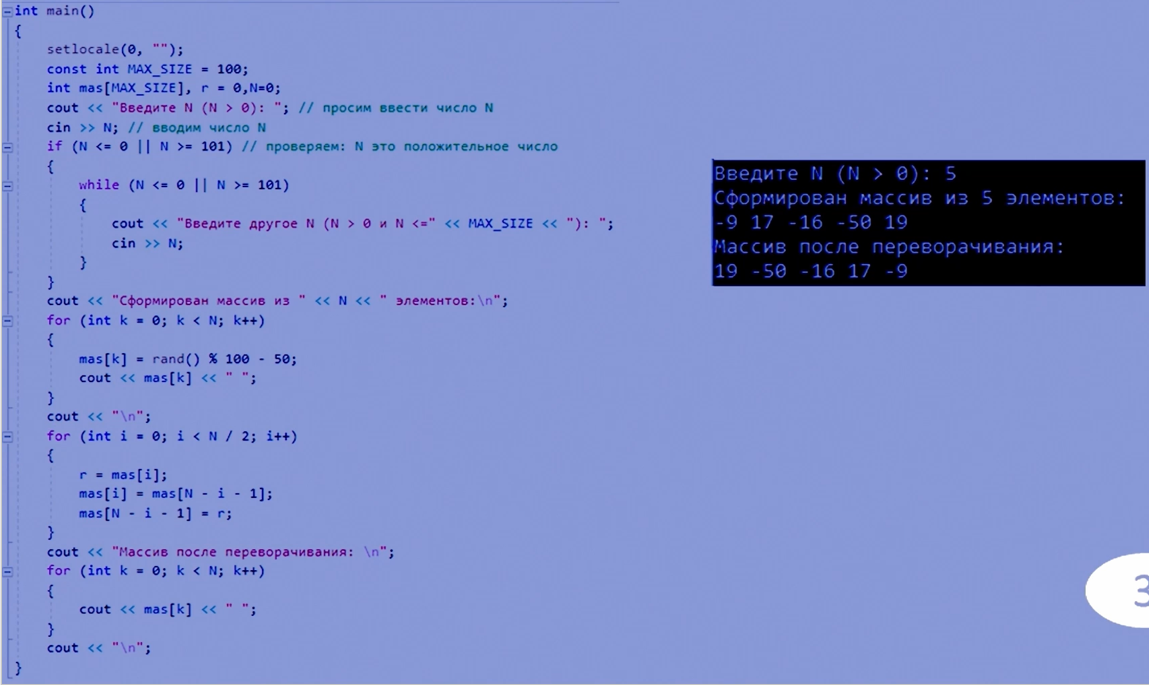


Рисунок 9.12 – Фрагмент кода к заданию 3

Четвертая задача: чтобы найти максимальный элемент и его индексы в двумерном массиве, нужно пройтись по всему массиву проверяя каждый элемент, больше ли он относительно максимального элемента. Если больше, то запоминаем его значение и индексы в отдельные переменные. В итоге получаем максимальный элемент и его расположение в двумерном массиве (рисунок 9.13).



Рисунок 9.13 - Фрагмент кода к заданию 4

Кроме статических массивов существуют и динамические. Динамическими называются массивы, размер которых может изменяться во время выполнения программы. Для выделения памяти под динамический массив на алгоритмическом языке C++ используется оператор «new», после которого в квадратных скобках указывается, сколько объектов будет содержать массив (рисунок 9.14).



Рисунок 9.14 – Пример объявления динамического массива

Динамические массивы можно создавать и с помощью функции «malloc». Функция malloc() в языке C++ выделяет память длиной для определенного количества байт и возвращает указатель на начало выделенной памяти (рисунок 9.15). Через полученный указатель можно помещать данные в выделенную память. Чтобы ее освободить, используется оператор free().

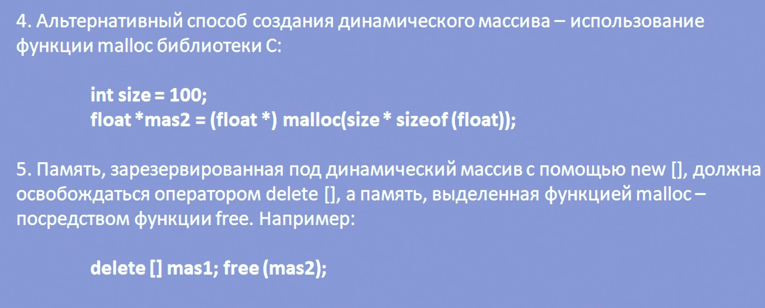


Рисунок 9.15 – Пример объявления динамического массива с помощью функции malloc()

После окончания работы с выделенной динамически памятью нужно освободить ее, чтобы она не занимала лишнее место в оперативной памяти. Для этой цели используется функция free() или delete[], которая возвращает память под управление операционной системы (рисунок 9.16).

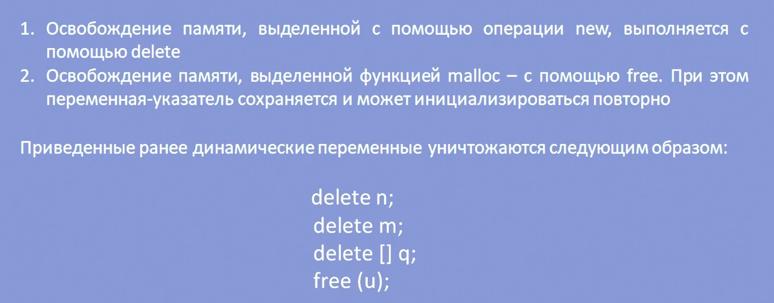


Рисунок 9.16 – Способы освобождения памяти динамического массива

Оператор delete необходимо использовать с квадратными скобками, чтобы не образовывался мусор в памяти компьютера. Если переменная-указатель выходит из области своего действия, отведенная под нее память освобождается. При этом память из-под самой динамической переменной не освобождается (рисунок 9.17).

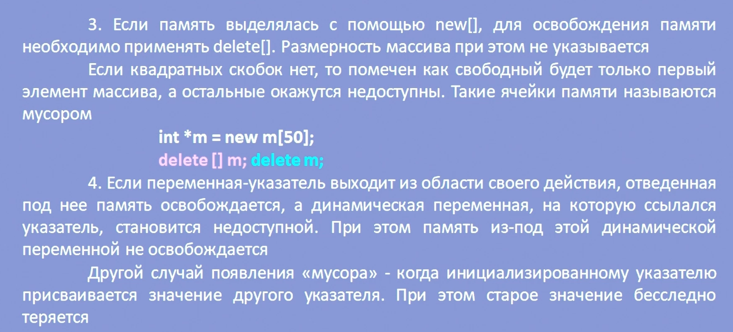


Рисунок 9.17 – Способы освобождения памяти динамического массива без появления мусора

Практические задачи:

1. Создать динамический одномерный массив целых чисел размерностью N, заполнить его случайными числами. Выполнить в данном массиве поиск заданного элемента, вывести его индекс и подсчитать количество совпадающих с ним элементов в исходном массиве.
2. Создать и заполнить динамический одномерный массив размерностью N. Удалить из полученного массива k элементов, начиная с конца.
3. Создать и заполнить динамический одномерный массив размерностью N. В полученный массив добавить m элементов, начиная с номера k. Результат работы программы вывести на экран.

Ответы к практическим задачам с пояснениями:

Для первой задачи создается динамический массив с введенной заранее размерностью. Далее он заполняется и вводится искомое число. Затем перебираются все элементы массива и проверяются на равенство с введенным значением. Их количество записывается в отдельную переменную (рисунок 9.18).



Рисунок 9.18 – Фрагмент кода к заданию 1

Вторая задача: для удаления k элементов с конца массива используется дополнительный динамический массив длиной N-k. Через цикл присваиваются значения элементов старого массива новому, затем удаляется старый массив (рисунок 9.19).

Если k = 0, то условие присваивания выполняться не будет, на выходе работы программы будет старый массив.

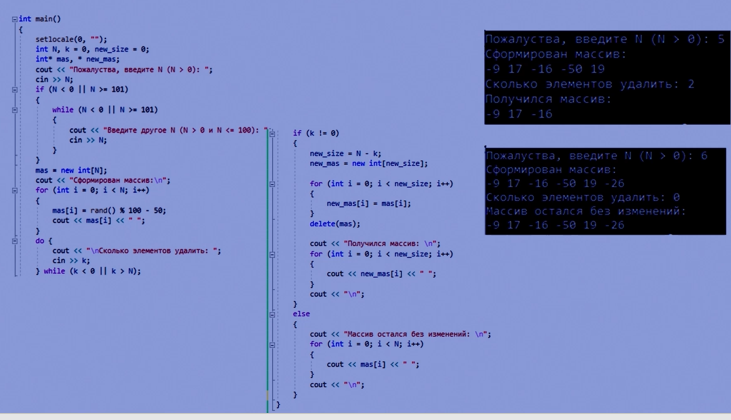


Рисунок 9.19 – Фрагмент кода к заданию 2

Третья задача: чтобы добавить в массив m элементов с номера k необходимо воспользоваться дополнительный динамическим массивом размерностью N + m. Первым циклом запишем в новый массив все элементы до элемента k. Затем введем m элементов после элемента k в новый массив вторым циклом. Третьим циклом запишем оставшиеся элементы старого массива (рисунок 9.20).

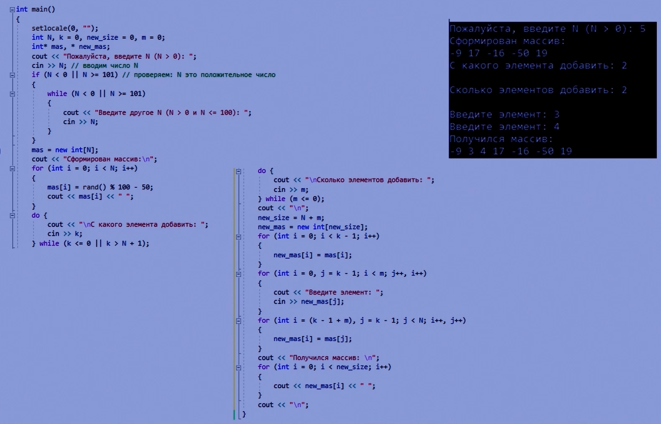


Рисунок 9.20 – Фрагмент кода к заданию 3

**9.2 Методы сортировок массивов**

*Метод вставки:*

1. Массив делится на две части. Выбирается один элемент из неотсортированной части и сохраняется в дополнительной переменной tmp
2. Отсортированная часть рассматривается с конца и элемент из tmp встает на свое место в этой части, не нарушая ее упорядоченность
3. Если место для tmp в отсортированной части найдено, то все элементы отсортированной последовательности, стоящие после этого места, сдвигаются вправо на один элемент. Элемент из tmp встает на найденное место
4. Размер отсортированной части увеличивается на один элемент
5. Процесс сортировки повторяется со второго шага до тех пор, пока не закончатся элементы в неотсортированной части.

На рисунке 9.21 приведен пример работы программы сортировки по возрастанию и убыванию.

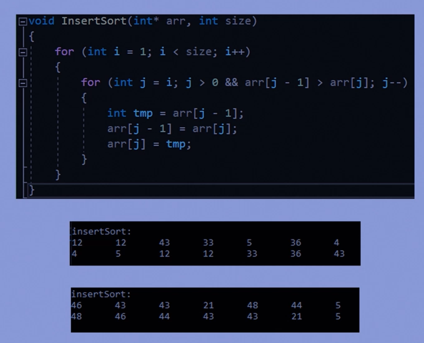


Рисунок 9.21 – Фрагмент кода сортировки методом вставки

*Метод выбора:*

1. Нахождение в массиве минимального либо максимального элемента и замена его местами с первым элементом массива.
2. Первый элемент исключается из дальнейшей обработки.
3. Нахождение в массиве минимального либо максимального элемента среди элементов с индексами от 1 до n-1 и замена его со вторым элементом массива.
4. Повторяется поиск минимального либо максимального элемента и его обмен с элементами со 2-го по n-1.

На рисунке 9.22 приведен пример работы программы сортировки методом выбора по возрастанию.



Рисунок 9.22 – Фрагмент кода сортировки методом выбора

*Метод обмена (сортировка пузырьком):*

1. Просматриваются 2 соседних элемента массива, двигаясь от его начала к концу.
2. Если левый элемент больше либо меньше правого, в зависимости в каком порядке требуется сортировка, то они меняются местами.
3. В результате наибольший либо наименьший элемент окажется на n-1 месте в массиве.
4. Так как на последнем месте максимальный либо минимальный элемент массива, то он исключается из дальнейшей обработки.
5. Первый и второй шаги повторяются до тех пор, пока массив не будет отсортирован.

На рисунке 9.23 приведен пример кода программы – сортировка методом обмена по возрастанию и убыванию.

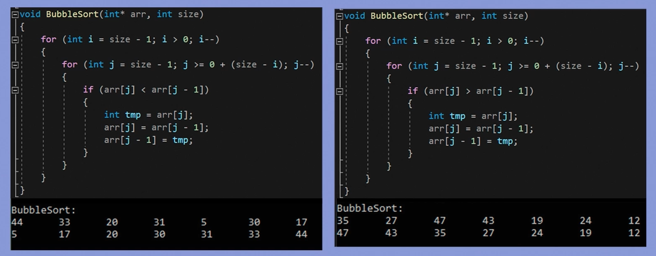


Рисунок 9.23 – Фрагмент кода сортировки методом обмена

Практические задачи:

1. Создать и заполнить одномерный массив. Отсортировать массив всеми тремя способами сортировки: методом вставки, методом выбора и методом обмена.
2. Создать и заполнить одномерный массив. Написать меню для работы с одномерным массивом на основе функций: создание массива, печать массива и сортировка массива любым из вышеупомянутых способов

Ответы с пояснениями к практическим задачам:

В первой задаче поочередно расписаны и выведены на экран с соответствующим оформлением коды трех ранее разобранных сортировок. Изначально на экран выводится неотсортированный массив со случайными значениями его элементов, затем он сортируется при помощи трех сортировок (рисунок 9.24).

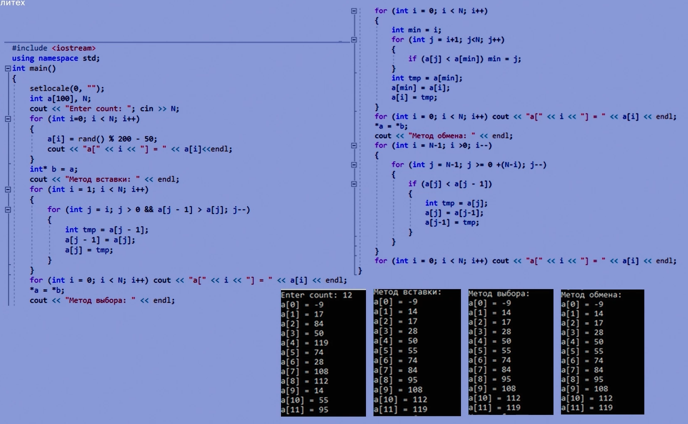


Рисунок 9.24 – Фрагмент кода к заданию 1

Во второй задаче поочередно разрабатываются функции для работы с массивом. Функция создания массива заполняет массив случайными значениями в промежутке от -50 до 49. Функция печати массива выводит каждый элемент массива с помощью цикла. Функция сортировки массива сортирует массив с помощью метода обмена. Меню написано с помощью цикла do {} while и выбор происходит на основе выполняющихся условий:

1. Создание нового массива.

2. Вывод значений элементов массива в консоль.

3. Выполнение сортировки элементов массива методом……

4. Завершение программы.

Также в программе предусмотрены различные проверки чтобы избежать некорректной работы или аварийного завершения программы (рисунок 9.25, рисунок 9.26).

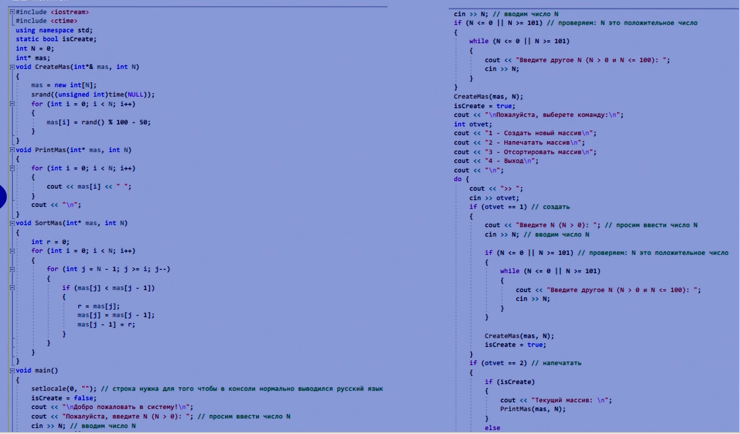


Рисунок 9.25 - Фрагмент кода к заданию 2

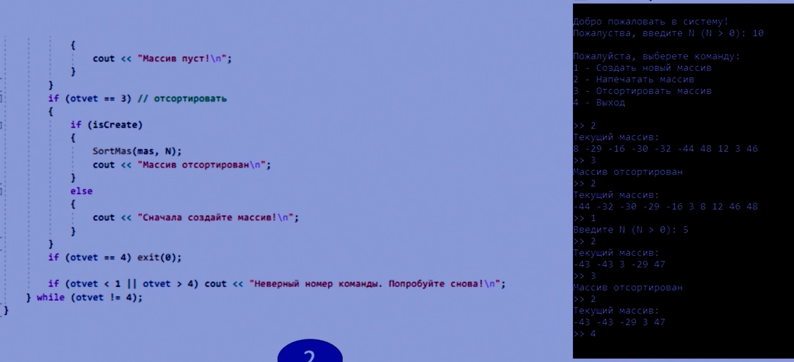


Рисунок 9.26 – Продолжение фрагмента кода к заданию 2