|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Одномерный массив»** | |
|  | |
| Выполнил студент группы ИКБО-09-22 | Гришин А. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Приобретение навыков по определению одномерного массива для структуры данных задачи, приобретение навыков создания алгоритмов операций над одномерным массивом и получение навыков по реализации алгоритмов операций над массивом через аппарат функций.

1. **Постановка задачи**
2. Разработать программу для выполнения операций на статическом массиве.
3. Разработать функции для ввода и вывода значений массива.
4. Выполнить декомпозицию задач, определенных вариантом индивидуального задания.
5. Разработать алгоритмы решения задач, определенных вариантом индивидуального задания.
6. Реализовать функции, выполняющие задачи, определенные вариантом индивидуального задания.
7. Разработать программу, демонстрирующую работу всех функций.
8. Изменить разработанную программу так, чтобы все операции выполнялись над динамическим массивом.
9. Изменить разработанную программу так, чтобы все операции выполнялись над динамическим массивом, используя контейнер <vector> для его представления в программе.
10. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Персональный вариант – 7.

Таблица 1. Задачи варианта.

|  |
| --- |
| * + - 1. Найти индекс элемента массива, цифровой корень которого равен 7.       2. Вставить новый элемент перед элементом, цифровой корень которого равен 7. Считать, что такое число одно.       3. Удалить элементы массива цифровой корень которых равен 7. |

# **Решение**

* 1. **Краткая теория по теме**

Массив – это линейная, однородная структура данных, состоящая из конечного числа элементов. Все элементы массива принадлежат одному типу данных. Доступ к элементу массива осуществляется по индексу. В языке программирования C++ можно выделить 3 типа хранилищ данных: статический массив, динамический массив и шаблон класса для контейнеров последовательностей библиотеки <vector>.

Статический массив – это структура данных, которая создается во время компиляции (т.е. массиву выделяется память). Размер массива нельзя изменять во время работы программы.

Формат определения одномерного статического массива:

|  |
| --- |
| тип имя\_массива[количество\_элементов]; |

Например:

|  |
| --- |
| const int size = 100;  int arr[size]; //для статического массива размер должен быть константой |

Операции можно выполнять только над элементом массива, допустимые базовым типом массива. Доступ к элементам массива осуществляется по индексу, например:

|  |
| --- |
| имя\_массива[индекс\_элемента] |

Каждый элемент занимает в памяти отдельную ячейку памяти. Ячейка памяти выделяется переменной, т.е. элемент массива – это отдельная переменная в составе массива.

Динамический массив – это структура данных, которая создается во время выполнения программы. Данный тип массива отличается от статического тем, что его изначальный размер можно задать в ходе выполнения программы. Динамический массив в языке С++ определяется через указатель. Для выделения динамической памяти под массив мы используем оператор new или функцию malloc().

После завершения работы с динамическим массивом мы может использовать функцию free(x) для освобождения памяти от использованных переменных, где x - указатель на массив:

|  |
| --- |
| int main()  {  int size = 10;  int arr = new int[size];  … // операции с массивом  free(arr);  return 0;  } |

Класс вектора стандартной библиотеки C++ — это шаблон класса для контейнеров последовательностей. Вектор хранит элементы заданного типа в линейном расположении и обеспечивает быстрый случайный доступ к любому элементу.

Для работы с вектором необходимо подключить заголовок <vector>:

|  |
| --- |
| #include <vector>  int main()  {  std::vector<int> arr;  return 0;  } |

Вектор является предпочтительным контейнером для последовательности, так как он удобнее и быстрее, чем тот же динамический массив.

* 1. **. Дополнительные функции**

Для упрощения работы над реализацией программы и повышения читаемости кода введем дополнительные функции, а именно для вывода массива и его заполнения отдельно для статического, динамического массивов и вектора.

|  |
| --- |
| template <typename ArrayType1>  void PrintArray(ArrayType1 arr, int size) {  for (int i = 0; i < size; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  void ArrayFilling(int\* arr, int size)  {  cout << "Заполните массив: ";  for (int i = 0; i < size; i++)  {  cin >> arr[i];  }  }  void ArrayFilling(vector<int>& arr, int size)  {  int num;  cout << "Заполните массив: ";  for (int i = 0; i < size; i++)  {  cin >> num;  arr.push\_back(num);  }  } |

* 1. **. Выполнение первой задачи. Найти индекс элемента массива, цифровой корень которого равен 7.**

Разберем реализацию выполнения первой задачи.

Для статического, динамического массивов и вектора задача реализуется одинаково с помощью универсального типа, заданного в шаблоне перед функцией. Универсальный тип может принимать любой другой тип данных, который не будет конфликтовать с работой функции. Так мы можем написать всего лишь одну функцию для любого типа массива и вектора. В данной функции программа вычисляет цифровой корень элемента массива.

|  |
| --- |
| int digitroot(int n) {  if (n < 10) {  return n;  }  else {  int s = 0;  while (n != 0) {  s += n % 10;  n /= 10;  }  return digitroot(s);  }  } |

После выполнения функция возвращает значение цифрового корня данного элемента массива. По условию нам необходимы элементы с цифровым корнем 7, поэтому мы делаем дополнительную проверку в основной части программы. Если элемент успешно проходит проверку, выводим его индекс. Если ни один элемент не подошёл по условию, выводим сообщение об ошибке поиска.

|  |
| --- |
| bool k = false;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  cout << "Цифровой корень равен 7 у элемента под индексом " << i;  cout << endl;  k = true;  }  }  if (k == false) {  cout << "Элементы с цифровым корнем 7 отсутствуют.";  cout << endl;  } |

* 1. **Выполнение второй задачи. Вставить новый элемент перед элементом, цифровой корень которого равен 7. Считать, что такое число одно.**

Для выполнения второй задачи нам уже потребуется не одна, а две функции для статического/динамического массива и вектора, так как в реализации для вектора необходимы функции из заголовка <vector>.

Рассмотрим реализацию на статическом массиве. Для начала благодаря функции определения цифрового корня мы находим индекс первого элемента, цифровой корень которого равен 7. Далее мы считываем новое число для вставки в массив. Далее программа вызывает функцию, в которой происходит вставка в массив нового числа перед элементом с цифровым корнем 7. В данную функцию мы передаем: сам массив, его размер для прохода по массиву, индекс элемента, найденный с помощью функции выше, и число, которое мы будем вставлять в массив.

В самой функции проход по массиву будет выполняться от большего индекса к меньшему (справа налево), так как, двигаясь к месту вставки, будет осуществляться постепенный сдвиг старших по индексу элементов на одну позицию вправо, тем самым расширяя массив. Если индекс элемента равен найденному индексу из первого задания, то значит перед данным элементом мы должны вставить новый элемент.

|  |
| --- |
| void pasteelement(int n, int x[], int num, int pos)  {  int i;  for (i = n; i > pos; i--) {  x[i] = x[i - 1];  }  x[pos] = num;  } |

После присвоения нужной ячейке значения введенного числа для вставки, функция завершает своё выполнение, и возвращает измененные массив и его размерность. Хоть функция и имеет возвращаемый тип void (ничего не возвращается), но мы передаем массив через указатель и размерность по ссылке, тем самым мы можем спокойно менять их значения, которые закрепятся за ними и в вызывающей функции.

Для динамического массива процесс вставки происходит также, так как мы используем тот же метод и передаем в него массив через указатель. Однако при работе с динамическим массивом нам необходимо контролировать количество выделяемой динамической памяти под хранение данных, поэтому воспользуемся функцией realloc() для выделения нужного количества динамической памяти.

|  |
| --- |
| x = (int\*) realloc(x, sizeof(int) \* (ARRSIZE + 1)); |

Аргументами функции realloc() являются указатель на ранее выделенный блок памяти, в нашем случае массив, и новый размер в байтах.

Для реализации на векторе будем использовать перегрузку функции для статического и динамического массивов, так в ней используется функция вектора emplace(), которая вставляет элемент в определенную ячейку вектора, причем остальные ячейки сдвигаются на одну позицию вправо.

|  |
| --- |
| vector<int> vectorpasteelement(vector<int> x, int num, int pos)  {  x.emplace(x.begin() + pos, num);  return x;  } |

В самой функции мы также увеличиваем размерность вектора для корректного отображения вектора в меню, и передаём в функцию emplace() позицию и сам элемент для вставки. Так как мы передали вектор и размерность по ссылке, то их значение меняется за пределами вызываемой функции. Данная функция будет возвращать нам векторный массив в изменённом виде.

* 1. **Выполнение третьей задачи. Удалить элементы массива цифровой корень которых равен 7.**

Для статического и динамического массивов реализация будет выполняться с помощью одной функции, как для первой и второй задачи. В неё мы передаем массив, размер массива и индекс, найденный в первой задаче, элемента, цифровой корень которого равен 7.

В данной функции мы уменьшаем значение размерности массива на единицу и сдвигаем элементы влево, начиная с элемента, индекс которого мы передали в функцию, и заканчивая последним элементом массива. Так мы производим удаление путём замещения удаляемого элемента соседним, старшим по индексу.

|  |
| --- |
| void deleteelement(int x[], int n, int pos)  {  for (int i = pos; i < n; ++i)  {  x[i] = x[i + 1];  }  } |

Со статическим всё происходит как и прежде, но у динамического опять надо изменить количество выделяемой динамической памяти, поэтому используем функцию realloc() с уменьшенным количеством байтов.

|  |
| --- |
| x = (int\*)realloc(x, sizeof(int) \* ARRSIZE-k); |

Используем мы её после операции удаления с уже уменьшенным размером, так как если уменьшить выделяемую память до вызова функции, то памяти будет недостаточно, так как массив еще содержит удаляемый элемент.

Для вектора мы также, как и во второй задаче, используем перегрузку функции для статического и динамического массивов. В ней мы изменяем размерность массива, уменьшая её каждый раз на единицу, когда находим элемент с цифровым корнем 7, и используем для вектора функцию erase(), которая использует в качестве аргумента позицию удаляемого элемента в векторе. Данная функция будет возвращать нам векторный массив в изменённом виде.

|  |
| --- |
| vector<int> vectordeleteelement(vector<int> x)  {  bool k = true;  while (k) {  k = false;  for (int i = 0; i < x.size(); i++) {  if (digitroot(x[i]) == 7) {  x.erase(x.begin() + i);  k = true;  break;  }  }  }  return x;  } |

* 1. **Описание работы программы и её интерфейса**

При запуске программы пользователь выбирает, что будет использоваться для хранения данных и выполнения над ними операций. На выбор у пользователя есть: статический массив, динамический массив и вектор. После выбора определенного пункта, программа просит ввести изначальную размерность для хранилища данных (в случае динамического и векторного массивов), либо размерность будет равна 10 (если массив является статическим). Далее рассмотрим реализацию отдельно для каждого пункта.

Если пользователь выбрал статический, то создается массив размерностью 10, Далее массив заполняется пользователем 10 элементами. После создания массива пользователю выводится меню операций над массивом, которые уже были описаны.

При выборе динамического массива, программа создает массив с помощью операции new, где в качестве аргумента принимает будущую размерность массива. В случае с вектором, программа создает вектор без фиксированной размерности, а переменная размера массива нужна лишь для корректного вывода вектора.

Далее представлен интерфейс программы (рис. 1).

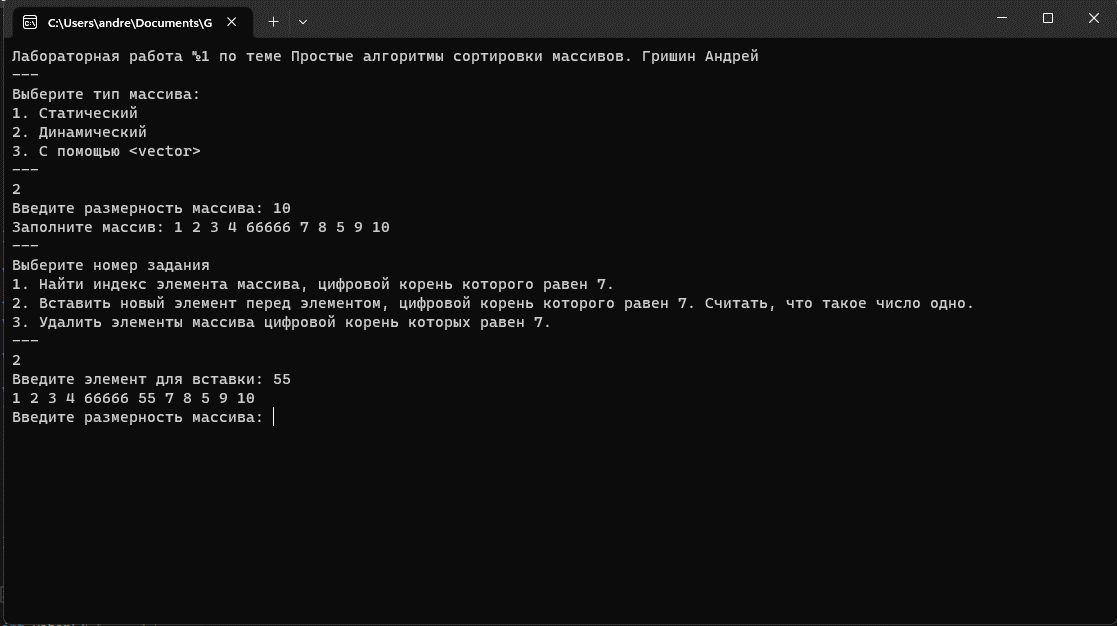


Рисунок 1. Интерфейс программы

1. **Тестирование**

Протестируем работу программы для всех трёх задач на статическом, динамическом массивах и векторе по отдельности.

Рассмотрим выполнение первой задачи на статическом массиве (рис. 2), динамическом массиве (рис. 3), векторе (рис. 4) и при отсутствии искомого элемента (рис. 5).

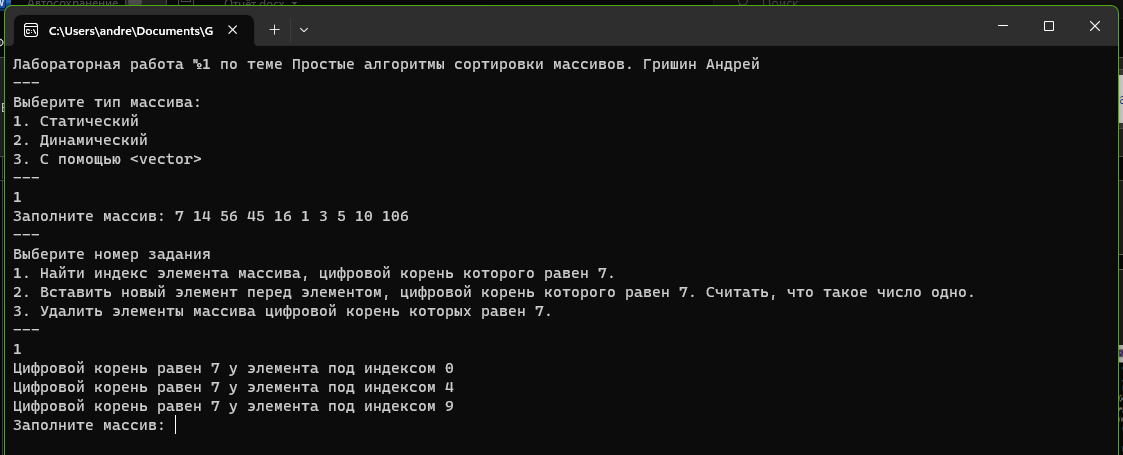


Рисунок 2. Тестирование первой задачи на статическом массиве

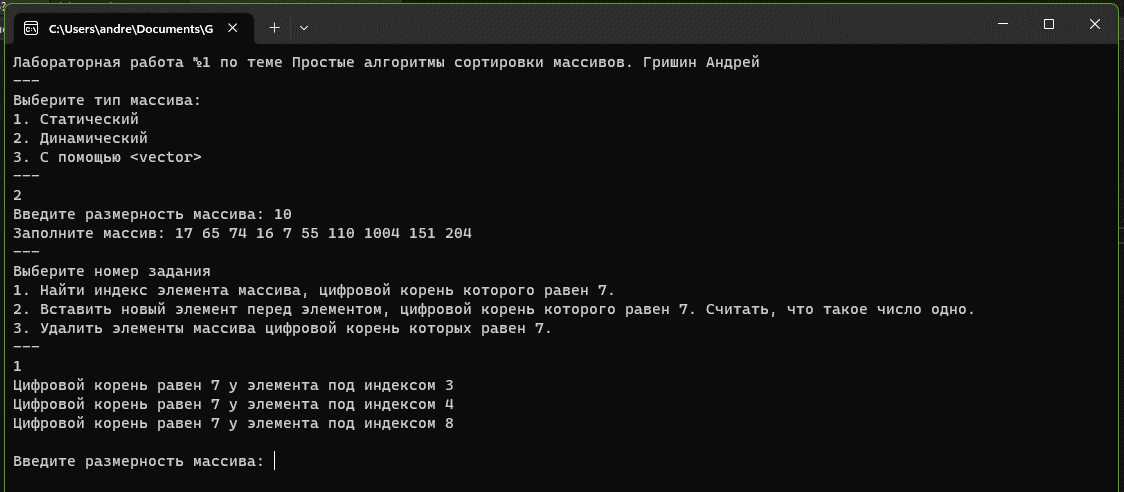


Рисунок 3. Тестирование первой задачи на динамическом массиве

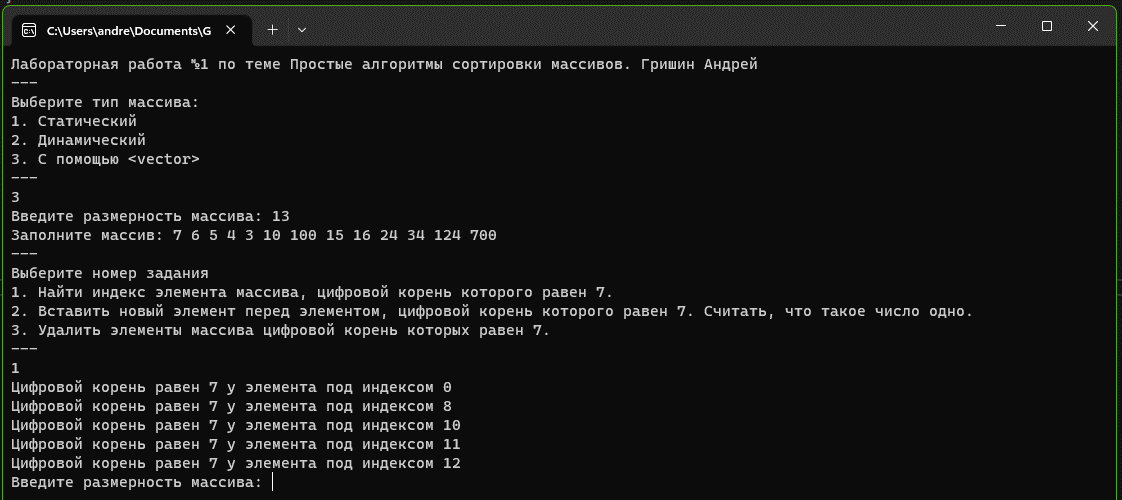


Рисунок 4. Тестирование первой задачи на векторе

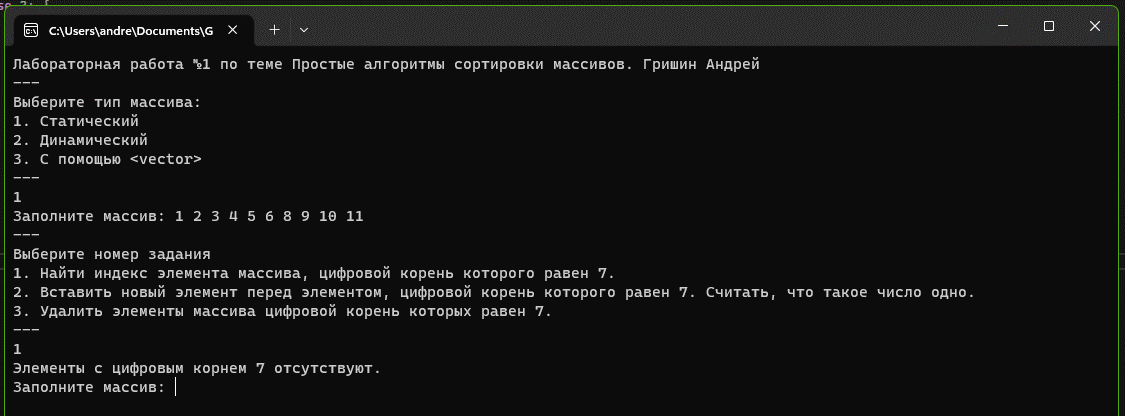


Рисунок 5. Тестирование первой задачи

при отсутствии искомого элемента

Тестирование первой задачи пройдено успешно.

Теперь протестируем выполнение второй задачи на статическом массиве (рис. 6), динамическом массиве (рис. 7), векторе (рис. 8) и при отсутствии нужного элемента (рис. 9), индекс которого мы находим в первой задаче.

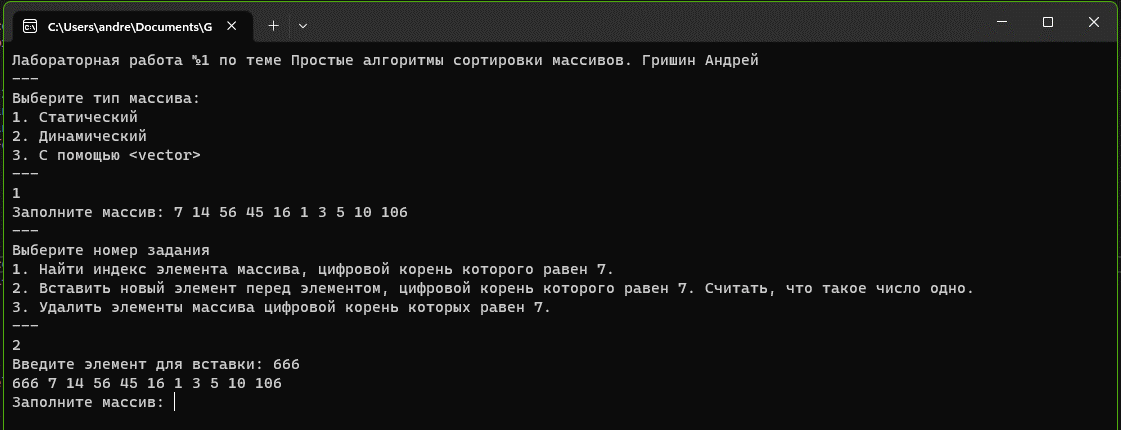


Рисунок 6. Тестирование второй задачи на статическом массиве

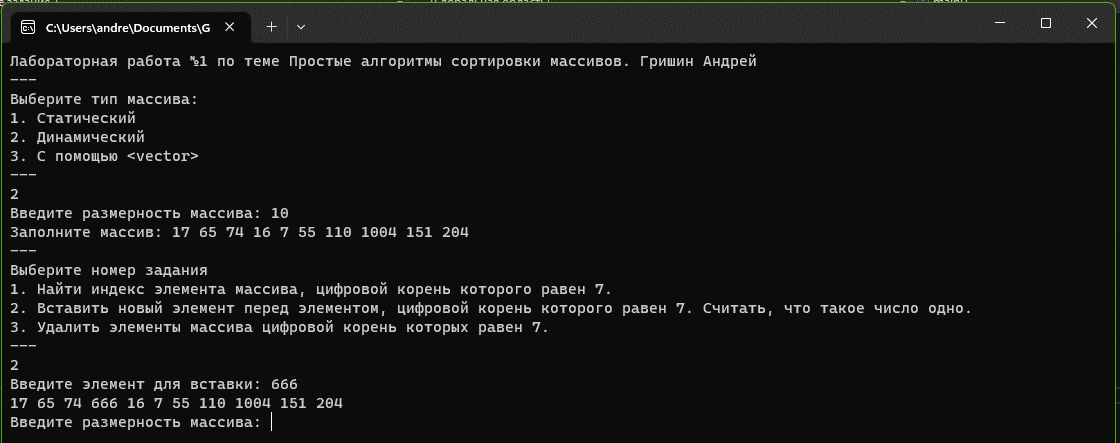


Рисунок 7. Тестирование второй задачи на динамическом массиве

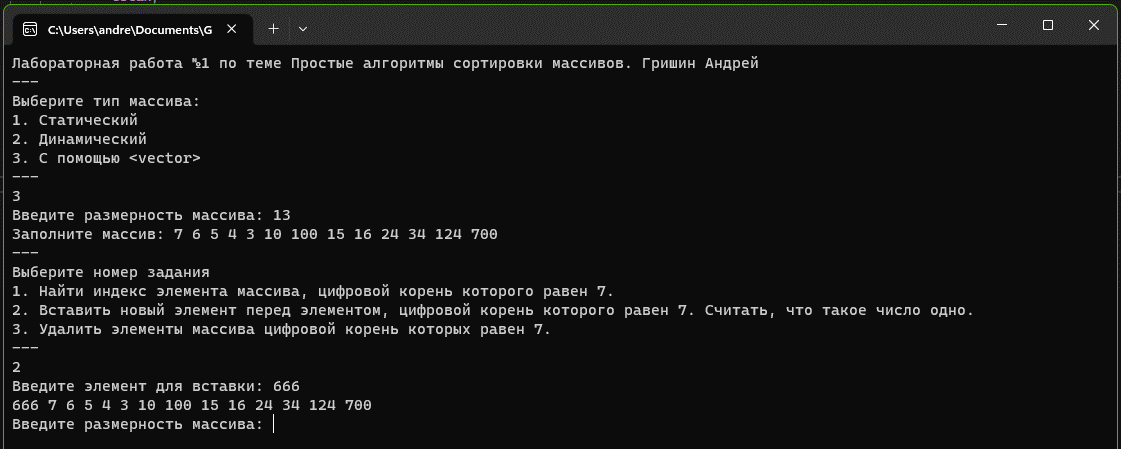


Рисунок 8. Тестирование второй задачи на векторе

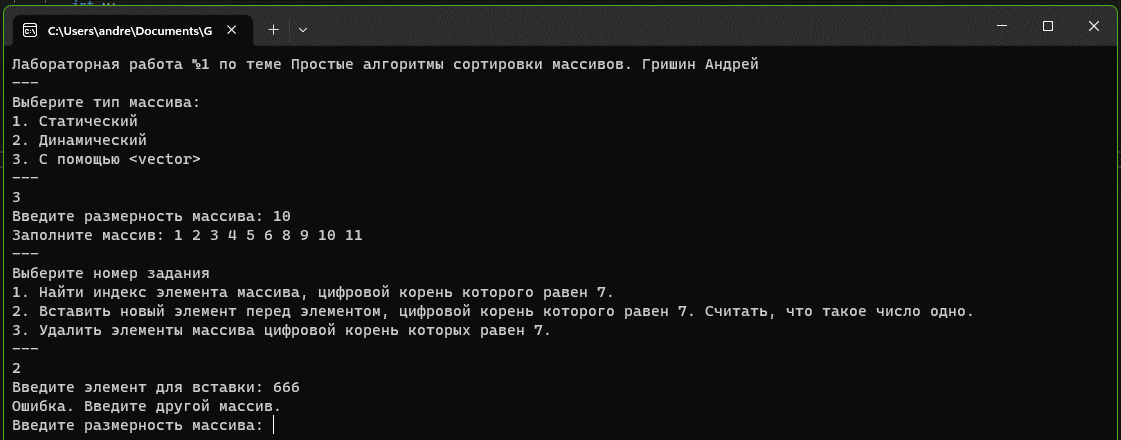


Рисунок 9. Тестирование второй задачи при отсутствии числа,

имеющего цифровой корень, равный 7

Тестирование второй задачи пройдено успешно.

Последним пройдет тестирование третьего задания на тех же статическом (рис. 10), динамическом (рис. 11) массивах, векторе (рис. 12) и при отсутствии нужного для выполнения задания элемента массива (рис. 13).

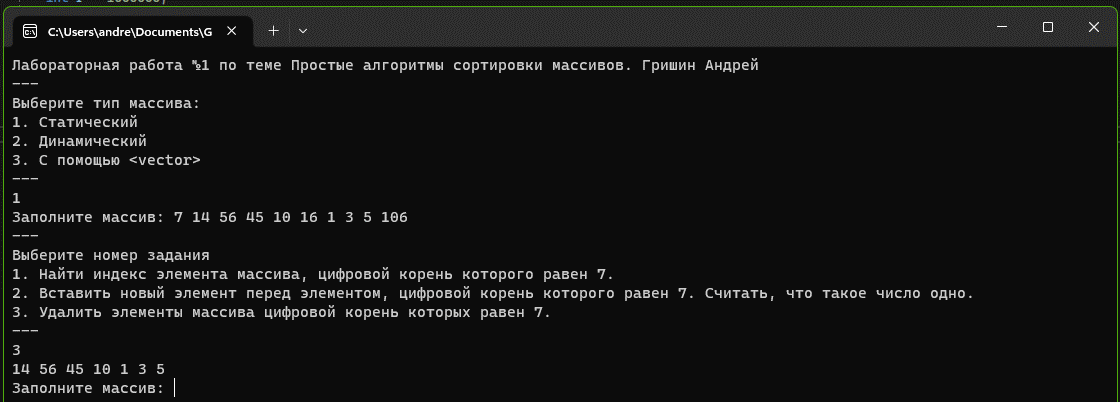


Рисунок 10. Тестирование третей задачи на статическом массиве

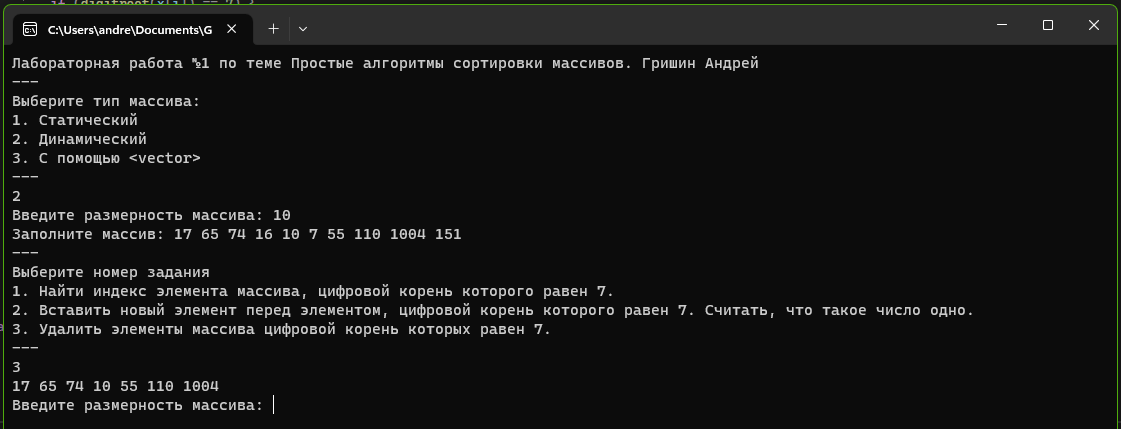


Рисунок 11. Тестирование третей задачи на динамическом массиве

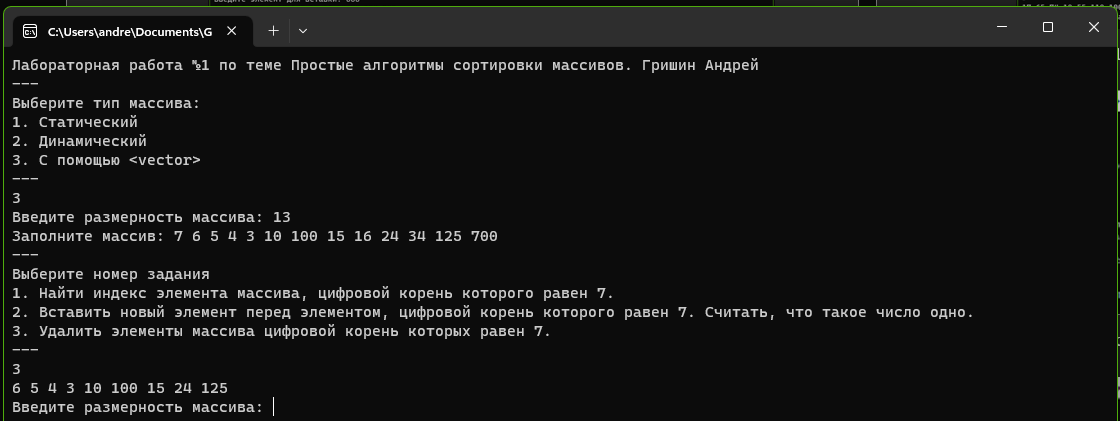


Рисунок 12. Тестирование третей задачи на векторе

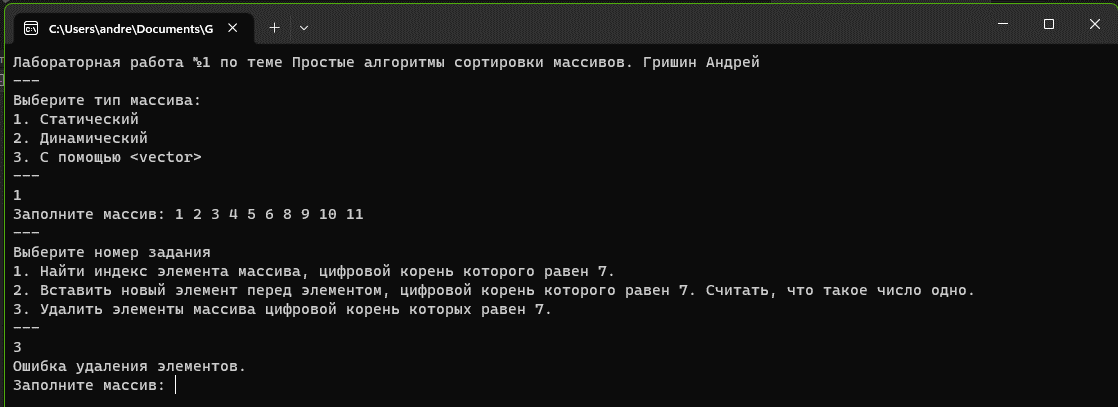


Рисунок 13. Тестирование третей задачи при отсутствии числа,

цифровой корень которого равен 7

Тестирование третей задачи пройдено успешно. Так мы протестировали все случаи для всех трёх задач, результат которых при определенных входных данных даёт желаемый результат.

1. **Вывод**

В результате выполнения работы я приобрел навыки по определению одномерного массива для структуры данных задачи и для создания алгоритмов операций над одномерным массивом в языке программирования C++.

1. **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  int digitroot(int n) {  if (n < 10) {  return n;  }  else {  int s = 0;  while (n != 0) {  s += n % 10;  n /= 10;  }  return digitroot(s);  }  }  void pasteelement(int n, int x[], int num, int pos)  {  int i;  for (i = n; i > pos; i--) {  x[i] = x[i - 1];  }  x[pos] = num;  }  vector<int> vectorpasteelement(vector<int> x, int num, int pos)  {  x.emplace(x.begin() + pos, num);  return x;  }  vector<int> vectordeleteelement(vector<int> x)  {  bool k = true;  while (k) {  k = false;  for (int i = 0; i < x.size(); i++) {  if (digitroot(x[i]) == 7) {  x.erase(x.begin() + i);  k = true;  break;  }  }  }  return x;  }  void deleteelement(int x[], int n, int pos)  {  for (int i = pos; i < n; ++i)  {  x[i] = x[i + 1];  }  }  template <typename ArrayType1>  void PrintArray(ArrayType1 arr, int size) {  for (int i = 0; i < size; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  void ArrayFilling(int\* arr, int size)  {  cout << "Заполните массив: ";  for (int i = 0; i < size; i++)  {  cin >> arr[i];  }  }  void ArrayFilling(vector<int>& arr, int size)  {  int num;  cout << "Заполните массив: ";  for (int i = 0; i < size; i++)  {  cin >> num;  arr.push\_back(num);  }  }  int vibor0() {  cout << "---" << endl;  cout << "Выберите тип массива:\n1. Статический\n2. Динамический\n3. С помощью <vector>" << endl;  cout << "---" << endl;  int n;  cin >> n;  return n;  }  int vibor()  {  int n;  cout << "---" << endl;  cout << "Выберите номер задания" << endl;  cout << "1. Найти индекс элемента массива, цифровой корень которого равен 7." << endl;  cout << "2. Вставить новый элемент перед элементом, цифровой корень которого равен 7. Считать, что такое число одно." << endl;  cout << "3. Удалить элементы массива цифровой корень которых равен 7." << endl;  cout << "---" << endl;  cin >> n;  return n;  }  int main()  {  setlocale(LC\_ALL, "RU");  cout << "Лабораторная работа №1 по теме Простые алгоритмы сортировки массивов. Гришин Андрей" << endl;  switch (vibor0()) {  case 1: {  while (true) {  const int ARRSIZE = 10;  int x[ARRSIZE];  ArrayFilling(x, ARRSIZE);  switch (vibor()) {  case 1: {  bool k = false;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  cout << "Цифровой корень равен 7 у элемента под индексом " << i;  cout << endl;  k = true;  }  }  if (k == false) {  cout << "Элементы с цифровым корнем 7 отсутствуют.";  cout << endl;  }  break;  }  break;  case 2: {  int y;  int r = 1000000;  int k = 0;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  if (i < r) {  r = i;  }  k++;  }  }  if (k == 0) {  r = 0;  }  cout << "Введите элемент для вставки: "; cin >> y;  if (r != 1000000) {  pasteelement(ARRSIZE, x, y, r);  int d = ARRSIZE + 1;  PrintArray(x, d);  continue;  }  else {  cout << "Ошибка. Введите другой массив." << endl;  }  break;  }  break;  case 3: {  int k = 0;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  deleteelement(x, ARRSIZE, i);  k++;  }  }  if (k != 0) {  PrintArray(x, ARRSIZE - k);  }  else {  cout << "Ошибка удаления элементов." << endl;  }  break;  }  }  }  }  break;  case 2: {  while (true)  {  int ARRSIZE;  cout << "Введите размерность массива: ";  cin >> ARRSIZE;  int\* x = new int[ARRSIZE];  ArrayFilling(x, ARRSIZE);  switch (vibor()) {  case 1: {  bool k = false;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  cout << "Цифровой корень равен 7 у элемента под индексом " << i << endl;  k = true;  }  }  if (k == false) {  cout << "Элементы с цифровым корнем 7 отсутствуют.";  }  cout << endl;  break;  }  break;  case 2: {  int y;  int r = 1000000;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  if (i < r) {  r = i;  }  }  }  cout << "Введите элемент для вставки: "; cin >> y;  if (r != 1000000) {  x = (int\*)realloc(x, sizeof(int) \* (ARRSIZE + 1));  pasteelement(ARRSIZE, x, y, r);  int d = ARRSIZE + 1;  PrintArray(x, d);  }  else {  cout << "Ошибка. Введите другой массив." << endl;  }  break;  }  break;  case 3: {  int k = 0;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  deleteelement(x, ARRSIZE, i);  k++;  }  x = (int\*)realloc(x, sizeof(int) \* ARRSIZE-k);  }  if (k != 0) {  PrintArray(x, ARRSIZE - k);  }  else {  cout << "Ошибка удаления элементов." << endl;  }  break;  }  }  }  }  break;  case 3: {  while (true)  {  int ARRSIZE;  cout << "Введите размерность массива: ";  cin >> ARRSIZE;  vector<int> x;  ArrayFilling(x, ARRSIZE);  switch (vibor()) {  case 1: {  bool k = false;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  cout << "Цифровой корень равен 7 у элемента под индексом " << i;  k = true;  cout << endl;  }  }  if (k == false) {  cout << "Элементы с цифровым корнем 7 отсутствуют.";  cout << endl;  }  break;  }  break;  case 2: {  int y;  int r = 1000000;  for (int i = 0; i < ARRSIZE; i++) {  if (digitroot(x[i]) == 7) {  if (i < r) {  r = i;  }  }  }  cout << "Введите элемент для вставки: "; cin >> y;  if (r != 1000000) {  x = vectorpasteelement(x, y, r);  PrintArray(x, x.size());  }  else {  cout << "Ошибка. Введите другой массив." << endl;  }  break;  }  break;  case 3: {  int k = 0;  for (int i = 0; i < ARRSIZE; i++) {  x = vectordeleteelement(x);  k++;  }  if (k != 0) {  PrintArray(x, x.size());  }  else {  cout << "Ошибка удаления элементов." << endl;  }  break;  }  }  }  }  break;  default: {  cout << "Вы ввели неверный номер.";  break;  }  }  } |