ЛАБОРАТОРНАЯ РАБОТА №1

Курс «Объектно-ориентированное программирование»

Тема: Синтаксис языка C++. Работа с памятью в C++.

Цель: Научиться программировать на языке C++ базовые операции с данными разных типов, разветвляющиеся и циклические алгоритмы, научиться работать с динамической памятью в C++, осуществлять консольный ввод-вывод данных.

Темы для предварительной проработки [УСТНО]**:**

- Среда разработки MS Visual Studio.
- Типы данных языка C++. Переменные, константы, функции.
- Арифметические операции, ветвление и циклы на языке C++.
- Консольный ввод-вывод (*iostream* и *cstdio*).
- Работа с памятью в C++. Указатели, ссылки, массивы, строки.

Индивидуальные задания [КОД] **:**

1. Написать программу обработки одномерного массива с *n* элементами, в соответствии с вариантом (*приложение А*). Необходимые действия должны производиться в функции **`processArray()`**, возвращающей определенное значение и формирующей еще один (выходной) массив данных (см. свой вариант). Память под массивы выделять статически (объявлять массивы локально в функции **`main`**) и для доступа к элементам использовать **индексы**. Ввод-вывод данных организовать средствами *cstdio*.
2. Написать программу, которая преобразует одномерный массив (1D) в двумерный (2D) (или наоборот), в соответствии с вариантом. Необходимо оформить в отдельных функциях код следующих действий: 1) инициализация массива; 2) вывод массива; 3) преобразование массива (создание нового массива с другой структурой). Память под массивы выделять динамически и для доступа к элементам использовать **указатели**. Ввод-вывод данных организовать средствами *iostream* и *iomanip*.
3. Написать свой аналог стандартной функции обработки строк из файла *cstring*, в соответствии с вариантом. В функции **`main`** на тестовых данных продемонстрировать результат работы как стандартной функции, так и собственной версии. Ввод-вывод данных организовать средствами *cstdio*.

Контрольные вопросы [ОТЧЕТ]**:**

1. Перечислите типы данных C++ с примерами объявления переменных.
2. Перечислите и кратко опишите классы памяти в C++.
3. Что из себя представляют указатели и ссылки? Приведите пример их взаимосвязи.
4. Что из себя представляет арифметика указателей (pointer arithmetic)?
5. Область действия переменных в C++. Приведите пример кода с memory leak.
6. Синтаксис функций на C++. Что такое сигнатура функции?
7. В чем заключается особенность переменных, объявленных в функциях с ключевым словом static?
8. Как можно организовать возврат значения в параметре функции?
9. Организация ввода-вывода с помощью cstdio.
10. Организация ввода-вывода с помощью iostream и iomanip.

Рекомендуемые источники:

[1] Страуструп Б. Язык программирования C++ / Б. Страуструп. – СПб.; М.: Невский диалект; Бином, 2015. – 1136 с.

[2] Лафоре Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – СПб.: Питер, 2013. – 928 с.

[3] Прата С. Язык программирования C++ / С. Прата; пер. с англ. Ю. Н. Артеменко. – Лекции и упражнения, 6-е изд.: Пер. с англ. – М.: Вильямс, 2012. – 1248 с.

[4] Дейтел Х. М. Как программировать на C++ / Х. М. Дейтел, П. Дж. Дейтел. – М.: Бином, 2000. – 1024 с.

Приложение А. Варианты индивидуальных заданий.

Вариант 0 (введен для примера, решение можно найти в приложении В)

1. Объявить массив из n=20 вещественных чисел, проинициализировать нулями. Функция `processArray()` должна заполнить массив случайными числами от -15.0 до 15.0, подсчитать и вернуть среднее значение average элементов массива и сформировать выходной булев массив, каждый элемент которого равен true (T), если соответствующий элемент входного массива больше среднего значения и false (F) в противоположном случае. Вывести на экран результирующие массивы.
   Пример:

   Вход: [-7 -6 13 -8 -10 -8 10 -10 -5 -3 -12 5 -13 -4 2 6 -10 -10 12 -5]
   Выход: [F F T F F F T F F T F T F F T T F F T F], average = -3

2. Преобразование: 1D ⟶ 2D. Одномерный массив из 20 целых чисел необходимо разложить по двумерной сетке 5x4 слева направо и сверху вниз.
   Инициализация: заполнить массив числами Фибоначчи: a[n] = a[n-1] + a[n-2].
   Вывод на экран: на каждый элемент массива отвести 7 позиций.

```
[1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765]

=>

[   1       1       2       3       5
    8      13      21      34      55
   89     144     233     377     610
  987    1597    2584    4181    6765  ]
```

3. Функция *strrchr*.
   Формат `char* strrchr(char* s, int c)`.
   Функция находит в строке s последнее вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Вариант 1

1. Объявить массив из n=15 вещественных чисел, проинициализировать единицами. Функция `processArray()` должна домножить все элементы с четными индексами на случайное число из диапазона [a, b] (a и b ввести с клавиатуры, a < 0), подсчитать и вернуть количество отрицательных элементов массива и сформировать выходной массив, содержащий только отрицательные элементы входного (т.е. размерность уменьшится). Вывести на экран результирующие массивы.
   Пример:

   Вход: [13 -6 13 12 -20 22 -10 10 -25 -13 -22 -25 17 6 12] a = -25 b= 25
   Выход: [-6 -20 -10 -25 -13 -22 -25], count = 7

2. Преобразование: 1D ⟶ 2D. Одномерный массив из 25 вещественных чисел необходимо разложить по двумерной сетке 5x5 слева направо и сверху вниз, но первый элемент на каждой строке должен содержать сумму остальных четырех.
   Инициализация: заполнить массив числами $x[n] = n \cdot \sin(\pi n/25)$, n – индекс элемента.
   Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[ 0 0.12533 0.49738 1.1044 1.927 2.9389 4.1073 5.3936 6.7546 8.1434 9.5106
10.805 11.976 12.974 13.752 14.266 14.477 14.354 13.869 13.006 11.756
10.117 8.0987 5.7199 3.008 ]

=>

[   3.6541   0.12533   0.49738    1.1044     1.927
    24.399    4.1073    5.3936    6.7546    8.1434
    49.508    10.805    11.976    12.974    13.752
    55.706    14.477    14.354    13.869    13.006
    26.943    10.117    8.0987    5.7199     3.008 ]
```

3. Функция *strchr*.
   Формат `char* strchr(char* s, int c)`.
   Функция находит в строке s первое вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Вариант 2

1. Объявить массив из n=20 вещественных чисел, проинициализировать нулями. Функция `processArray()` должна заполнить массив случайными числами от -20.0 до 70.0, а затем отнормировать массив, т.е. поделить каждый элемент массива на максимальное по модулю значение в массиве. Это значение она также должна вернуть на выходе. Сформировать выходной вещественный массив, в котором все элементы, стоящие до позиции максимального элемента включительно, повторяют элементы входного массива, а остальные равны x (число x ввести с клавиатуры). Вывести на экран результирующие массивы.
   Пример:

   Вход: [18 49 68 -13 15 -13 65 45 -10 -8 -17 0 42 -9 -3 61 15 -15 7 -10] x = 2.7

   Выход: max Item = 68
   [0.264706 0.720588 1 -0.191176 0.220588 -0.191176 0.955882 0.661765
   -0.147059 -0.117647 -0.25 0 0.617647 -0.132353 -0.0441176 0.897059 0.220588
   -0.220588 0.102941 -0.147059]

   [0.264706 0.720588 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7 2.7]

2. Преобразование: 2D ⟶ 1D. Двумерный массив 5x5 целых чисел необходимо выложить в один ряд по элементам слева направо и сверху вниз, исключая все элементы на четных строках.
   Инициализация: заполнить массив факториалами: x[i][j] = (i+j)! .
   Вывод на экран: на каждый элемент массива отвести 8 позиций.

```
 [       1        1        2        6       24
         1        2        6       24      120
         2        6       24      120      720
         6       24      120      720     5040
        24      120      720     5040    40320       ]

=>

[1       1       2       6       24       2       6       24      120
 720      24      120     720     5040    40320 ]
```

3. Функция *strcmp*.
   Формат `int strcmp(const char* s1, const char* s2)`.
   Функция сравнивает строку s1 со строкой s2 в лексикографическом порядке. Если s1<s2, то результат равен -1, если s1=s2, то 0, если s1>s2, то 1.

Вариант 3

1. Объявить массив из n=10 целых чисел, проинициализировать нулями. Функция `processArray()` должна заполнить массив членами геометрической прогрессии с начальным элементом b_1 и шагом q (ввести с клавиатуры), подсчитать и вернуть cnt – количество всех трехзначных элементов массива, а также сформировать выходной массив, который содержит все элементы исходного, кроме тех, которые делятся на 3 с остатком 1. Вывести на экран массивы.

   Вход: b1 = 2 q = 5 [2 10 50 250 1250 6250 31250 156250 781250 3906250]
   Выход: [2 50 1250 31250 781250] cnt = 1

2. Преобразование: 2D ⟶ 1D. Двумерный массив 4x4 вещественных чисел необходимо выложить в один ряд по элементам *справа налево* и сверху вниз.
   Инициализация: заполнить массив числами x[i][j] = √(i+j+1).
   Вывод на экран: каждый элемент одномерного массива вывести с точностью 4 знака после запятой; каждый элемент двумерного массива – с точностью 2 знака.

```
[    1   1.41   1.73      2
  1.41   1.73      2   2.24
  1.73      2   2.24   2.45
     2   2.24   2.45   2.65       ]

=>

[ 2 1.7321 1.4142 1 2.2361 2 1.7321 1.4142 2.4495 2.2361 2 1.7321 2.6458
  2.4495 2.2361 2 ]
```

3. Функция *strcat*.
   Формат `char* strcat(char* dest, const char* src)`.
   Функция приписывает строку src к строке dest.

Вариант 4

1. Объявить массив из n=20 целых чисел, проинициализировать нулями. Функция process_array() должна заполнить массив случайными числами от 1 до 10, вычислить и вернуть наиболее часто встречающееся значение в массиве (если таких несколько, вернуть наименьшее) и сформировать выходной массив из всех элементов, которые встречаются как минимум 2 раза во входном массиве. Вывести на экран массивы.

   Вход: [2 10 3 5 8 7 3 3 7 10 5 2 4 5 4 3 2 10 5 1]
   Выход: [2 3 4 5 7 10] MostFrequentElement = 3

2. Преобразование: 2D ⟶ 1D. Двумерный массив 5x3 вещественных чисел необходимо выложить в один ряд по элементам слева направо и *снизу вверх*.
   Инициализация: заполнить массив числами x[i][j] = sin(i–j) + cos(i+j).
   Вывод на экран: на каждый элемент массива отвести 14 позиций.

```
[            1        -0.3012         -1.325
         1.382        -0.4161         -1.831
        0.4932        -0.1485        -0.6536
       -0.8489         0.2557          1.125
         -1.41         0.4248          1.869          ]

=>

[ -1.41 0.4248 1.869 -0.8489 0.2557 1.125 0.4932 -0.1485 -0.6536 1.382
  -0.4161 -1.831 1 -0.3012 -1.325 ]
```

3. Функция *strncat*.
   Формат `char* strncat(char* dest, const char* src, size_t maxlen)`.
   Функция приписывает maxlen символов строки src к строке dest.

Вариант 5

1. Объявить массив из n=16 целых чисел, проинициализировать единицами. Функция `processArray()` должна заполнить элементы массива с *четными индексами* степенями двойки (1, 2, 4, 8, 16, …), с нечетными индексами – степенями тройки (3, 9, 27, …). Также подсчитать и вернуть count – количество двузначных чисел в массиве и сформировать выходной массив, содержащий только такие числа. Вывести на экран результирующие массивы.

   Вход: [3 2 9 4 27 8 81 16 243 32 729 64 2187 128 6561 256]
   Выход: [27 81 16 32 64] count = 5

2. Преобразование: 1D → 2D. Одномерный массив из 18 целых чисел необходимо отсортировать в порядке убывания и разложить по двумерной сетке 9x2 слева направо и сверху вниз.
   Инициализация: заполнить массив числами x[i] = i^2+1 и все элементы с четными индексами домножить на -1.
   Вывод на экран: на каждый элемент массива отвести 5 позиций.

```
[ 1 -2 5 -10 17 -26 37 -50 65 -82 101 -122 145 -170 197 -226 257 -290 ]

=>

[ 257   197
  145   101
   65    37
   17     5
    1    -2
  -10   -26
  -50   -82
 -122  -170
 -226  -290   ]
```

3. Функция *strcpy*.
   Формат `char* strcpy(char* dest, const char* src)`.
   Функция копирует строку src в строку dest.

Вариант 6

1. Объявить массив из n=15 вещественных чисел, проинициализировать единицами. Функция `processArray()` должна домножить все элементы с четными индексами на случайное число из диапазона [a, b] (a и b введите с клавиатуры, a < 0), подсчитать и вернуть количество двузначных элементов массива и сформировать выходной массив, содержащий только двузначные элементы входного (т. е. размерность уменьшится). Вывести на экран результирующие массивы.

   Вход:
   [13 -6 13 12 -20 22 -10 10 -25 -13 -22 -25 17 6 12] a = -25 b = 25

   Выход:
   [13 13 12 -20 22 -10 10 -25 -13 -22 -25 17 12]
   cnt = 13

2. Преобразование: 1D ⟶ 2D. Одномерный массив из 16 вещественных чисел необходимо разложить по двумерной сетке 4 x 4 слева направо и сверху вниз, но первый элемент на каждой строке должен содержать сумму остальных трех.
   Инициализация: заполнить массив числами x[n] = n · exp(πn/100), n – индекс элемента.
   Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[0 1.0319 2.1297 3.2965 4.5356 5.8504 7.2446 8.7218 10.286 11.941 13.691
 15.541 17.495 19.557 21.734 24.03]

                                    =>

[   6.4581    1.0319    2.1297    3.2965
    21.817    5.8504    7.2446    8.7218
    41.173    11.941    13.691    15.541
    65.321    19.557    21.734    24.03    ]
```

3. Функция *strncpy*.
   Формат `char* strncpy(char* dest, const char* src, size_t num)`.
   Функция копирует первые num символов из строки src в строку dest.

Вариант 7

1. Объявить массив из *n*=15 вещественных чисел, проинициализировать нулями. Функция `processArray()` должна заполнить массив случайными числами от 20.0 до 100.0, а затем вычесть из каждого элемента массива минимальное значение в массиве. Это значение она также должна вернуть на выходе. Сформировать выходной вещественный массив, в котором все элементы, стоящие до позиции минимального элемента включительно, повторяют элементы входного массива, а остальные равны минимальному элементу массива. Вывести на экран результирующие массивы.

   Вход:
   [38 79 68 29 65 43 85 93 72 80 37 23 31 92 83]

   Выход:
   min Item = 23

   [15 56 45 6 42 20 62 70 49 57 14 0 8 69 60]
   [38 79 68 29 65 43 85 93 72 80 37 23 23 23 23]

2. Преобразование: 2D ⟶ 1D. Двумерный массив 5x5 целых чисел необходимо выложить в один ряд по элементам слева направо и сверху вниз, исключая все элементы на нечетных строках.
   Инициализация: заполнить массив факториалами: x[i][j] = i! + j!.
   Вывод на экран: на каждый элемент массива отвести 8 позиций.

```
[       2       2       3       7      25
        2       2       3       7      25
        3       3       4       8      26
        7       7       8      12      30
       25      25      26      30      48       ]

                                =>

[2 2 3 7 25 7 7 8 12 30]
```

3. Функция *strchr*.
   Формат `char* strchr(char* s, int c)`.
   Функция находит в строке s первое вхождение символа c и возвращает подстроку, начинающуюся с этого символа.

Вариант 8

1. Объявить массив из n=15 целых чисел, проинициализируйте нулями. Функция `processArray()` должна заполнить массив членами арифметической прогрессии с начальным элементом a_1 и шагом d (ввести с клавиатуры), подсчитать и вернуть *cnt* – количество всех двузначных элементов массива, а также сформировать выходной массив, который содержит все элементы исходного, кроме тех, сумма цифр которых равна 10. Вывести на экран массивы.

   Вход:
   a1 = 7 d = 12
   [7 19 31 43 55 67 79 91 103 115 127 139 151 163 175]

   Выход:
   [7 31 43 67 79 103 115 139 151 175]
   cnt = 7

2. Преобразование: 2D → 1D. Двумерный массив 4x4 вещественных чисел необходимо выложить в один ряд по элементам слева направо и снизу вверх.
   Инициализация: заполнить массив числами x[i][j] = √(i + j + 1).
   Вывод на экран: каждый элемент одномерного массива вывести с точностью четыре знака после запятой; каждый элемент двумерного массива – с точностью два знака.

```
[    1   1.41   1.73      2
  1.41   1.73      2   2.24
  1.73      2   2.24   2.45
     2   2.24   2.45   2.65     ]

                                  =>

[2 2.2361 2.4495 2.6458 1.7321 2 2.2361 2.4495 1.4142 1.7321 2 2.2361 1 1.4142
 1.7321 2]
```

3. Функция *strncat*.
   Формат `char* strncat(char* dest, const char* src, size_t maxlen)`.
   Функция приписывает maxlen символов строки src к строке dest.

Вариант 9

1. Объявить массив из *n*=20 целых чисел, проинициализировать нулями. Функция `processArray()` должна заполнить массив случайными числами от одного до пяти, вычислить и вернуть наименее часто встречающееся значение в массиве (если таких несколько, вернуть наименьшее из них) и сформировать выходной массив из всех элементов входного, отсортированных в порядке возрастания. Вывести на экран массивы.

   Вход:
   [1 3 2 2 1 5 4 2 3 1 2 3 1 2 5 4 3 3 2 4]
   Выход:
   [1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 4 4 4 5 5] Least_frequent_element = 5

2. Преобразование: 2D → 1D. Двумерный массив 5x3 вещественных чисел необходимо выложить в один ряд по элементам *по столбцам сверху вниз*.
   Инициализация: заполнить массив числами x[i][j] = sin(i – j) + cos(i + j).
   Вывод на экран: на каждый элемент массива отвести 10 позиций.

```
[            1        -0.3012         -1.325
         1.382        -0.4161         -1.831
        0.4932        -0.1485        -0.6536
       -0.8489         0.2557          1.125
         -1.41         0.4248          1.869          ]

                                      =>

[1 1.382 0.4932 -0.8489 -1.41 -0.3012 -0.4161 -0.1485 0.2557 0.4248 -1.325 -
 1.831 -0.6536 1.125 1.869]
```

3. Функция *strcat*.
   Формат `char* strcat(char* dest, const char* src)`.
   Функция приписывает строку src к строке dest.

Вариант 10

1. Объявить массив из *n*=16 целых чисел, проинициализировать единицами. Функция `processArray()` должна заполнить элементы массива с *четными индексами* степенями двойки (1, 2, 4, 8, 16,…), с нечетными индексами – степенями тройки (3, 9, 27,…). Также подсчитать и вернуть *cnt* – количество чисел в массиве, содержащих цифру 1, и сформировать выходной массив, являющийся отсортированной версией входного. Вывести на экран результирующие массивы.

   Вход:
   [3 2 9 4 27 8 81 16 243 32 729 64 2187 128 6561 256]
   Выход:
   [2 3 4 8 9 16 27 32 64 81 128 243 256 729 2187 6561]
   cnt = 5

2. Преобразование: 1D → 2D. Одномерный массив из 18 целых чисел необходимо отсортировать в порядке убывания и разложить по двумерной сетке 6x3 слева направо и сверху вниз по столбцам.
   Инициализация: заполнить массив числами x[i] = i^2 – 1 и все элементы с нечетными индексами домножить на –1.
   Вывод на экран: на каждый элемент массива отвести 5 позиций.

```
[1 0 -3 8 -15 24 -35 48 -63 80 -99 120 -143 168 -195 224 -255 288]

                                    =>

[   1   -35  -143
    0    48   168
   -3   -63  -195
    8    80   224
  -15   -99  -255
   24   120   288   ]
```

3. Функция *strncpy*.
   Формат `char* strncpy(char* dest, const char* src, size_t num)`.
   Функция копирует первые num символов из строки src в строку dest.

Приложение В. Фрагменты кода для варианта 0.

Задание 1.
В главной функции создаем статические массивы и вызываем функцию **`processArray()`**.

```
int main()
{
	// размер массива равен 20 (по заданию)
	const size_t N = 20;

	double a[N] = { 0.0 };	// создаем массив и инициализируем нулями
	bool b[N] = { false };	// булев массив - "false-ми"

	// инициализация генератора случайных чисел
	srand(0);
	// подсчет среднего значения в массиве a, изменение массива b
	double result = processArray(a, N, b);
					// вывод массивов на экран
	printArray(a, N);
	printBoolArray(b, N);
					// ...и среднего значения элементов массива a
	printf("Average: %g\n\n", result);
	return 0;
}
```

В функции `processArray()` заполняем массив случайными числами, считаем среднее и формируем массив `flags` булевых значений. Обратите внимание на сигнатуру: запись `double arr[]` эквивалентна записи `double* arr` (т.е. передаче параметра по указателю).

```
double processArray(double arr[], size_t n, bool flags[])
{
	for (size_t i = 0; i < n; ++i)	// заполнение массива случ. числами [-15.0..15.0]
	{
		arr[i] = (double)(rand() % 30) - 15.0;
	}

	double average = 0.0;

	for (size_t i = 0; i < n; ++i)		// подсчет среднего значения элементов
	{
		average += arr[i];
	}
	average /= n;

	for (size_t i = 0; i < n; ++i)		// формирование булевого массива (по заданию)
	{
		flags[i] = arr[i] > average;
	}

	return average;			// возвращаем среднее
}
```

Пример функции вывода на экран всех значений булевого массива:

```
void printBoolArray(bool* arr, size_t n)
{
	for (size_t i = 0; i < n; ++i)
	{
		(arr[i]) ? printf("T ") : printf("F ");
	}
	printf("\n");
}
```

Задание 2.
В главной функции создаем динамически входной одномерный массив `int* arr=new int [LEN]`. Двумерный массив создаем из одномерного массива `arr` в функции `makeArray2D(arr, N, M)`.

```
int main()
{
	const size_t LEN = 20;			// задаем все параметры
	const size_t N = 5;
	const size_t M = 4;

	int* arr = new int [LEN];			// выделяем память под входной массив

	initializeArray(arr, LEN);			// инициализируем массив (по варианту)

	printArray1D(arr, LEN);			// выводим на консоль массив

	// конвертируем 1-мерный массив в 2-мерный (по варианту)
	int** arr2D = makeArray2D(arr, N, M);

	printArray2D(arr2D, N, M);			// выводим на консоль 2-мерный массив-результат

	freeArray1D(arr);				// освобождаем память 1-мерного массива
	freeArray2D(arr2D, N);			// освобождаем память 2-мерного массива
	return 0;
}
```

Код функции `int** makeArray2D(int* arr1D, size_t rows, size_t cols)` может быть таким:

```
int** makeArray2D(int* arr1D, size_t rows, size_t cols)
{
	int** arr2D = new int* [rows];		// выделяем память под выходной массив

	for (size_t i = 0; i < rows; ++i)		// и еще под каждый ряд массива в отдельности
	{
		*(arr2D + i) = new int [cols];
	}

	for (size_t i = 0; i < rows; ++i)
	{
		for (size_t j = 0; j < cols; ++j)
		{
			*(*(arr2D + i) + j) = *(arr1D + i*cols + j);
		}
	}

	return arr2D;		// возвращаем указатель на выделенную память под 2-мерный массив
}
```

По заданию, здесь работа с массивами производится через указатели.
Запись

```
*(*(arr2D + i) + j) = *(arr1D + i*cols + j)
```

эквивалентна более понятной «индексной» записи

```
arr2D[i][j] = arr1D[i*cols + j]
```

Код инициализации массива может выглядеть, например, так:

```
void initializeArray(int* arr, size_t n)
{
	if (n > 0)
	{
		*arr = 1.0;
	}

	if (n > 1)
	{
		*(arr + 1) = 1.0;
	}

	for (size_t i = 2; i < n; ++i)
	{
		*(arr + i) = *(arr + i-1) + *(arr + i-2);
		// эквивалентно: arr[i] = arr[i-1] + arr[i-2]
	}
}
```

Остальные функции `printArray1D`, `printArray2D`, `freeArray1D`, `freeArray2D` предлагаются студенту на самостоятельную проработку.

Задание 3.

В главной функции объявляем тестовую строку и вызываем поочередно стандартную `strrchr` и свою версию – `reverseChar`.

```
int main()
{
	// тестовая строка
	char s[] = "www.some_address.and_something_else.com";

	// проверяем стандартную функцию (ищем последнее вхождеие символа ".")
	char* standard_result = strrchr(s, '.');

	// проверяем свою версию стандартной функции
	char* my_result = reverseChar(s, '.');

	// выводим на экран строку,
	// затем результат стандартной функции
	// и затем результат своей версии
	printf("Initial string: %s\nStandard: %s\nMy version: %s\n",
                                                    s, standard_result, my_result);
	return 0;
}


char* reverseChar(char* s, int c)
{
	size_t len = 0;              // стартовый индекс

	char* rest_of_string = NULL; // указатель на оставшуюся часть строки (его нужно вернуть)

	// проходим до конца строки (пока не встретим символ 0)
	while (s[len] != '\0')
	{
		// при этом если очередной символ оказался равен искомому c,
		if (s[len] == c) rest_of_string = s + len;  // то запоминаем указатель на него
		len++;
	}
	// возвращаем последний присвоенный указатель
	return rest_of_string;
}
```

Вообще, возможны разные алгоритмы, приводящие к одним и тем же результатам. Например, вот еще пример возможной реализации (менее эффективной, почему?) той же функции strrchr() :

```
char* reverseChar2(char* s, int c)
{
	// стартовый индекс
	int len = 0;

	// проходим до конца строки (пока не встретим символ 0)
	while (s[len] != '\0') len++;

	// затем идем обратно, пока не найдем искомый символ
	while (s[len] != c && len >= 0) len--;

	// возвращаем указатель на часть строки, начиная с найденной позиции
	return s + len;
}
```

Наконец, третий вариант через указатели. Например, функция *strrchr()* может быть реализована на языках C/C++ таким образом (более компактная версия функции *reverseChar*, приведенной выше):

```
const char* strrchr(const char* s, int c)
{
	const char* ret = 0;
	do
	{
		if (*s == (char)c)  ret = s;
	}
	while (*s++);

	return ret;
}
```