МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

**отчет**

по практической работе №3

по дисциплине «Программирование»

Тема: "Указатели и многомерные статические массивы"

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0323 |  | Рудницкая Т.С. |
| 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.

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. В C++ существует три вида указателей:

Указатели на объект, который содержит адрес области памяти, хранящей данные определенного типа.

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.

Указатели на void. Такой указатель применяется в тех случаях, когда тип объекта, адрес которого нужно хранить, не определен. Указателю типа void 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 &.

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

1. #include using namespace std;
2. int main() {
3. setlocale(LC\_ALL, "Russian");
4. float pi = 3.14, \*p; p = &pi;
5. cout << "Значение переменной pi: " << pi << endl;
6. cout << "Значение переменной pi: " << \*p << endl;
7. cout << "Адрес переменной pi: " << p << endl;
8. return 0;
9. Значение переменной pi: 3.14
10. Значение переменной pi: 3.14
11. Адрес переменной pi: 012FF77C

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.

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.

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.

Постановка задачи.

Необходимо написать программу, которая:

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.



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



Используя арифметику указателей, сортирует элементы любой сортировкой.

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

Выполнение работы.

Работа представлена с помощью графического интерфейса C++/CLI. Так как неизвестно, какую размерность матрицы выберет пользователь, в работе используется динамический массив. При закрытии формы, массив (а также временный массив в четверть размера основного) уничтожаются командой delete[].

Так как из-за особенностей отрисовки формы, не получилось сделать паузы при выведении ячеек массива на экран, массив заполняется не случайными числами, а по порядку.

Заполнение массива спиралью:

1. void Coursach::Lab3::fillArrSpiral()
2. {
3. int counter = 0;
4. int i = 0;
5. int j = 0;
6. //проверяем, пока счётчик не станет равен количеству ячеек массива
7. while (counter < arrayWidth \* arrayWidth) {
8. //присваиваем значение текущей ячейке
9. //\*(pArr[i] + j) = getRandomNumber(matrixMin, matrixMax);
10. \*(pArr[i] + j) = counter;
11. showNumber(i, j, 0);
12. //проверяем, куда надо двигаться
13. //если представить массив в виде таблицы, у которой есть четыре стороны
14. //двигаемся вправо, если текущий элемент ещё не достиг правого края массива (или заполненного столбца)
15. if (i <= j + 1 && i + j < arrayWidth - 1)
16. ++j;
17. //двигаемся вниз, если мы находимся у правого края таблицы, но ещё не достигли правого нижнего угла
18. else if (i < j && i + j >= arrayWidth - 1)
19. ++i;
20. //двигаемся влево, если достигли нижнего угла
21. else if (i >= j && i + j > arrayWidth - 1)
22. --j;
23. //в противном случае двигаемся вверх
24. else
25. --i;
26. //подсчитываем заполненные элементы
27. counter++;
28. //всё повторяется, только "стенки" таблицы сдвигаются, благодаря первой проверке в каждом из условий
29. }
30. }

Заполнение массива змейкой:

1. void Coursach::Lab3::fillArrSnake(){
2. int counter = 0;
3. int i = 0;
4. int j = 0;
5. //проверяем, пока счётчик не станет равен количеству ячеек массива
6. while (counter < arrayWidth \* arrayWidth) {
7. //присваиваем значение текущей ячейке, используя арифметику указателей

\*(pArr[i] + j) = counter;

1. showNumber(i, j, 0);
2. //проверяем, если столбец нулевой или чётный, и текущий элемент ещё не достиг низа, то передвигаемся вниз
3. if (j % 2 == 0 && i < arrayWidth - 1)
4. i++;
5. //если столбец нечётный, и текущий элемент ещё не достиг верха, о сдвигаемся вверх
6. else if (j % 2 != 0 && i > 0)
7. i--;
8. //или меняем столбец
9. else
10. j++;
11. counter++;
12. }
13. }

Чтобы переставить блоки матрицы, условно делю матрицы на «верхние» и «нижние» или «правые» и «левые» четверти (в зависимости от схемы). Единственная схема, для которой будет недостаточно сделать std::swap между нужными четвертями, это схема номер 1, в которой блоки переставляются по кругу. Чтобы избежать затирания информации, одну из четвертей нужно скопировать во временный массив, остальные сдвинуть по кругу, и вставить скопированную четверть на своё место. Я выбрала верхнюю левую, а в конце операции она, соответственно, вставляется на место верхней правой четверти.

Функции для копирования во временный массив и вставки на постоянное место при перестановке по кругу:

1. void Coursach::Lab3::copyFirstQuarter() {
2. for (int i = 0; i < arrayWidth / 2; i++)
3. for (int j = 0; j < arrayWidth / 2; j++)
4. \*(pArrTemp[i] + j) = \*(pArr[i] + j);
5. }
6. void Coursach::Lab3::pasteFirstQuarter() {
7. for (int i = 0; i < arrayWidth / 2; i++)
8. for (int j = 0; j < arrayWidth / 2; j++)
9. std::swap(\*(pArrTemp[i] + j), \*(pArr[i] + j + arrayWidth / 2));
10. }

Функции для перестановки четвертей по вертикали, горизонтали и диагонали в двух направлениях:

1. void Coursach::Lab3::setElsLeftToRight(int minI) {
2. for (int i = minI; i < minI + arrayWidth / 2; i++)
3. for (int j = 0; j < arrayWidth / 2; j++)
4. std::swap(\*(pArr[i] + j), \*(pArr[i] + j + arrayWidth / 2));
5. }
6. void Coursach::Lab3::setElsUpToDown(int minJ) {
7. for (int i = 0; i < arrayWidth / 2; i++)
8. for (int j = minJ; j < minJ + arrayWidth / 2; j++)
9. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j));
10. }
11. void Coursach::Lab3::setElsDiagonalLeftToRight() {
12. for (int i = 0; i < arrayWidth / 2; i++)
13. for (int j = 0; j < arrayWidth / 2; j++)
14. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j + arrayWidth / 2));
15. }
16. void Coursach::Lab3::setElsDiagonalRightToLeft() {
17. for (int i = 0; i < arrayWidth / 2; i++)
18. for (int j = arrayWidth / 2; j < arrayWidth; j++)
19. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j - arrayWidth / 2));
20. }

Для перестановки блоков матриц нужно просто вызвать эти функции в нужном порядке.

Сортировка двумерного динамического массива сложнее, чем сортировка двумерного статического массива, потому что внутренние массивы не стоят в памяти по порядку. Даже простейшая сортировка пузырьком в двумерном динамическом массиве, ещё и с использованием арифметики указателей выполняется непросто, хотя в итоге алгоритм выходит несложный.

Код сортировки:

1. void Coursach::Lab3::insertSort()
2. {
3. int i, j, k, jp, ip;
4. int size = arrayWidth \* arrayWidth;
5. int flag;
6. do {
7. flag = 0;
8. for (k = 1; k < size; k++) {
9. //Вычисляем индексы текущего элемента
10. i = k / arrayWidth;
11. j = k - i \* arrayWidth;
12. //Вычисляем индексы предыдущего элемента
13. ip = (k - 1) / arrayWidth;
14. jp = (k - 1) - ip \* arrayWidth;
15. if (\*(pArr[i] + j) < \*(pArr[ip] + jp)) {
16. std::swap(\*(pArr[i] + j), \*(pArr[ip] + jp));
17. flag = 1;
18. }
19. }
20. } while (flag);
21. }

Действия над матрицей производятся просто перебиранием элементов по очереди во вложенном цикле. Так как размеры матрицы ограничены, при умножении и делении результаты округляются до третьего знака.

1. void Coursach::Lab3::addition(float num){
2. for (int i = 0; i < arrayWidth; i++) {
3. for (int j = 0; j < arrayWidth; j++)
4. \*(pArr[i] + j) = \*(pArr[i] + j) + num;
5. }
6. }
7. void Coursach::Lab3::subtraction(float num)
8. {
9. for (int i = 0; i < arrayWidth; i++) {
10. for (int j = 0; j < arrayWidth; j++)
11. \*(pArr[i] + j) = \*(pArr[i] + j) - num;
12. }
13. }
14. void Coursach::Lab3::multiple(float num)
15. {
16. for (int i = 0; i < arrayWidth; i++) {
17. for (int j = 0; j < arrayWidth; j++)
18. \*(pArr[i] + j) = round(\*(pArr[i] + j) \* num \* 1000) / 1000;
19. }
20. }
21. void Coursach::Lab3::division(float num)
22. {
23. for (int i = 0; i < arrayWidth; i++) {
24. for (int j = 0; j < arrayWidth; j++)
25. \*(pArr[i] + j) = round(\*(pArr[i] + j)/num \* 1000)/1000;
26. }
27. }

Полный код программы размещён в приложении А.

Выводы.

Самым сложным заданием для меня оказалась сортировка. В процессе выполнения работы я закрепила всё, что знала об указателях и узнала много нового о двумерных динамических массивах.

Приложение А

Полный код программы

Файл lab3.h

1. //служебные функции, нужные для первого запуска
2. /\*\*
3. \* инициализация, получение значений необходимых констант
4. \*/
5. private: void initialise();
6. /\*\*
7. \* расстановка количества строк и колонок в матрице
8. \*/
9. private: void matrixSetup();
10. //Функции проверки пользотельского выбора
11. /\*\*
12. \* Проверка, выбран ли метод заполнения матрицы
13. \* если да, то функция возвращает код метода
14. \*/
15. private: int getFillMethod();
16. /\*\*
17. \* Проверка, выбран ли порядок матрицы
18. \* если да, то функция возвращает порядок
19. \*/
20. private: int getMatrixDegree();
21. /\*\*
22. \* Проверка, выбран ли порядок матрицы
23. \* если да, то функция возвращает порядок
24. \*/
25. //private: int getMatrixTrans();
26. /\*\*
27. \* Проверка введённого юзером числа
28. \* Принимает текст поля ввода
29. \*/
30. private: float checkUserNum(String^ text);
31. /\*\*
32. \*вывод ошибки
33. \*/
34. private: void showError(String^ text);
35. //действия над матрицей
36. /\*\*
37. \* Подготовка матрицы к новому действию:
38. \* вывод текущего состояния матрицы слева
39. \* показывание матрицы справа, если она была спрятана
40. \*/
41. private: void prepareMatrix();
42. /\*\*
43. \* Прячем всё лишнее (ошибки) с формы
44. \*/
45. private: void hideError();
46. /\*\*
47. \* вывод массива
48. \* {@params} {matrixNum}: выбор, какую из матриц показывать
49. \* 0 - левая матрица
50. \*/
51. private: void showArray(int matrixNum);
52. /\*\*
53. \* вывод элемента массива
54. \* {@params} {i, j}: координаты элемента
55. \* {matrixNum}: выбор, какую из матриц показывать
56. \* 0 - левая матрица
57. \*/
58. private: void showNumber(int i, int j, int matrixNum);
59. /\*\*
60. \* Заполнение матрицы, согласно выбранной схемы
61. \*/
62. private: void selectScheme();
63. /\*\*
64. \* Заполнение матрицы спиралью
65. \*/
66. private: void fillArrSpiral();
67. /\*\*
68. \* Заполнение матрицы змейкой
69. \*/
70. private: void fillArrSnake();
71. //Замена четвертей матрицы
72. //Набор функций для схемы перестановки блоков
73. /\*\*
74. \* копирование первой четверти матрицы во временный массив
75. \* нужно для того, чтобы во время замены четвертей матрицы по кругу не потерялся один из кусков
76. \*/
77. void copyFirstQuarter();
78. /\*\*
79. \* вставка скопированной части на своё место в правый верхний уго исходной матрицы
80. \*/
81. void pasteFirstQuarter();
82. /\*\*
83. \* функция свапа левой и правой четвертей
84. \* @param {minI} - начиная с какого номера строки, отсчитатьчетверть, которую нужно поменять местами с соседней
85. \*/
86. void setElsLeftToRight(int minI);
87. /\*\*
88. \* функция свапа верхней и нижней четвертей
89. \* @param {minJ} - начиная с какого номера столбца, отсчитатьчетверть, которую нужно поменять местами с соседней
90. \*/
91. void setElsUpToDown(int minJ);
92. /\*\*
93. \* функция свапа по-диагонали
94. \* верхний левый угол с правым нижним
95. \*/
96. void setElsDiagonalLeftToRight();
97. /\*\*
98. \* функция свапа по-диагонали
99. \* верхний правый угол с левым нижним
100. \*/
101. void setElsDiagonalRightToLeft();
102. /\*\*
103. \* замена четвертей матрицы по кругу
104. \* копируется верхняя левая четверть, потом меняются местами верхняя левая с нижней левой
105. \* потом нижняя левая с нижней правой
106. \* потом нижняя правая с верхней правой
107. \* и, наконец, вставка в верхнюю правую четверть сохранённого куска матрицы
108. \*/
109. void roundShift();
110. /\*\*
111. \* замена четвертей матрицы по-диагонали
112. \*/
113. void diagonalShift();
114. /\*\*
115. \* замена верхней половины матрицы на нижнюю и обратно
116. \*/
117. void verticalShift();
118. /\*\*
119. \* замена левой половины матрицы на правую и обратно
120. \*/
121. void horizontalShift();
122. /\*\*
123. сортировка вставками заданного массива
124. @param {\*p} - указатель на массив, в который будутзаписыватьсязначения
125. \*/
126. void insertSort();
127. /\*\*
128. \* сложение элементов матрицы с числом, введённым пользователем
129. \* @param {num} - число, введённое пользователем
130. \*/
131. //Арифметические действия над матрицей
132. void addition(float num);
133. /\*\*
134. \* вычитание из элементов матрицы числа, введённым пользователем
135. \* @param {num} - число, введённое пользователем
136. \*/
137. void subtraction(float num);
138. /\*\*
139. \* умножение элементов матрицы на число, введённым пользователем
140. \* @param {num} - число, введённое пользователем
141. \*/
142. void multiple(float num);
143. /\*\*
144. \* деление элементов матрицы на число, введённым пользователем
145. \* @param {num} - число, введённое пользователем
146. \*/
147. void division(float num);
148. //События формы
149. //закрытие формы
150. private: System::Void btnCloseForm\_Click\_1(System::Object^ sender, System::EventArgs^ e);
151. //Генерация массива
152. private: System::Void btnGetArray\_Click(System::Object^ sender, System::EventArgs^ e);
153. //Сортировка массива
154. private: System::Void btnSort\_Click(System::Object^ sender, System::EventArgs^ e);
155. private: System::Void btnAdd\_Click(System::Object^ sender, System::EventArgs^ e);
156. private: System::Void btnSub\_Click(System::Object^ sender, System::EventArgs^ e);
157. private: System::Void btnMtpl\_Click(System::Object^ sender, System::EventArgs^ e);
158. private: System::Void btnDiv\_Click(System::Object^ sender, System::EventArgs^ e);
159. private: System::Void quotersOptionsCombo\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {
160. if (btnQuotersChange->Visible == false) btnQuotersChange->Visible = true;
161. }
162. private: System::Void btnQuotersChange\_Click(System::Object^ sender, System::EventArgs^ e);

Файл lab3.cpp:

1. #include "Lab3.h"
2. float\*\* pArr, \*\* pArrTemp;
3. int arrayWidth;
4. int fillMethod;
5. int matrixMin, matrixMax, matrixCols, matrixRows;
6. int errors = 0;
7. System::Void Coursach::Lab3::btnCloseForm\_Click\_1(System::Object^ sender, System::EventArgs^ e)
8. {
9. delete[] pArr;
10. delete[] pArrTemp;
11. this->Close();
12. }
13. void Coursach::Lab3::initialise()
14. {
15. pArr = new float\* [arrayWidth];
16. pArrTemp = new float\* [arrayWidth / 2];
17. for (int i = 0; i < arrayWidth; i++)
18. pArr[i] = new float[arrayWidth];
19. for (int i = 0; i < arrayWidth / 2; i++)
20. pArrTemp[i] = new float [arrayWidth / 2];
21. matrixMin = 0;
22. matrixMax = arrayWidth \* arrayWidth;
23. matrixCols = arrayWidth;
24. matrixRows = arrayWidth;
25. }
26. void Coursach::Lab3::matrixSetup()
27. {
28. matrixView->RowCount = matrixRows;
29. matrixView->ColumnCount = matrixCols;
30. matrixView2->RowCount = matrixRows;
31. matrixView2->ColumnCount = matrixCols;
32. }
33. void Coursach::Lab3::showError(String^ text)
34. {
35. errorLabel1->Text += text + "\n";
36. errorPanel->Visible = true;
37. errors += 1;
38. }
39. void Coursach::Lab3::hideError()
40. {
41. errorPanel->Visible = false;
42. errorLabel1->Text = "";
43. }
44. void Coursach::Lab3::prepareMatrix()
45. {
46. errors = 0;
47. hideError();
48. showArray(0);
49. }
50. int Coursach::Lab3::getMatrixDegree()
51. {
52. if (getMatrixDegreeCombo->SelectedIndex == -1) {
53. showError("Выберите порядок матрицы");
54. return -1;
55. }
56. return arrayWidth = Convert::ToInt16(getMatrixDegreeCombo->SelectedItem);
57. }
58. int Coursach::Lab3::getFillMethod() {
59. if (getStyleCombo->SelectedIndex == -1) {
60. showError("Выберите стиль заполнения!");
61. return -1;
62. }
63. return getStyleCombo->SelectedIndex;
64. }
65. void Coursach::Lab3::selectScheme()
66. {
67. switch (fillMethod) {
68. case 0:
69. fillArrSpiral();
70. break;
71. case 1:
72. fillArrSnake();
73. break;
74. }
75. }
76. void Coursach::Lab3::showArray(int matrixNum)
77. {
78. for (int i = 0; i < arrayWidth; i++)
79. for (int j = 0; j < arrayWidth; j++)
80. showNumber(i, j, matrixNum);
81. if (secondMatrixPanel->Visible == false) matrixView2->Visible = true;
82. }
83. void Coursach::Lab3::showNumber(int i, int j, int matrixNum)
84. {
85. if (matrixNum == 0) {
86. matrixView->Rows[i]->Cells[j]->Value = pArr[i][j];
87. }
88. else {
89. matrixView2->Rows[i]->Cells[j]->Value = pArr[i][j];
90. }
91. }
92. void Coursach::Lab3::fillArrSpiral()
93. {
94. int counter = 0;
95. int i = 0;
96. int j = 0;
97. //проверяем, пока счётчик не станет равен количеству ячеек массива
98. while (counter < arrayWidth \* arrayWidth) {
99. //присваиваем значение текущей ячейке
100. //\*(pArr[i] + j) = getRandomNumber(matrixMin, matrixMax);
101. \*(pArr[i] + j) = counter;
102. showNumber(i, j, 0);
103. //проверяем, куда надо двигаться
104. //если представить массив в виде таблицы, у которой есть четыре стороны
105. //двигаемся вправо, если текущий элемент ещё не достиг правого края массива (или заполненного столбца)
106. if (i <= j + 1 && i + j < arrayWidth - 1)
107. ++j;
108. //двигаемся вниз, если мы находимся у правого края таблицы, но ещё не достигли правого нижнего угла
109. else if (i < j && i + j >= arrayWidth - 1)
110. ++i;
111. //двигаемся влево, если достигли нижнего угла
112. else if (i >= j && i + j > arrayWidth - 1)
113. --j;
114. //в противном случае двигаемся вверх
115. else
116. --i;
117. //подсчитываем заполненные элементы
118. counter++;
119. //всё повторяется, только "стенки" таблицы сдвигаются, благодаря первой проверке в каждом из условий
120. }
121. }
122. void Coursach::Lab3::fillArrSnake()
123. {
124. int counter = 0;
125. int i = 0;
126. int j = 0;
127. //проверяем, пока счётчик не станет равен количеству ячеек массива
128. while (counter < arrayWidth \* arrayWidth) {
129. //присваиваем значение текущей ячейке, используя арифметику указателей
130. //\*(pArr[i] + j) = getRandomNumber(matrixMin, matrixMax);
131. \*(pArr[i] + j) = counter;
132. showNumber(i, j, 0);
133. //Sleep(100);
134. //проверяем, если столбец нулевой или чётный, и текущий элемент ещё не достиг низа,
135. //то передвигаемся вниз
136. if (j % 2 == 0 && i < arrayWidth - 1)
137. i++;
138. //если столбец нечётный, и текущий элемент ещё не достиг верха,
139. //то сдвигаемся вверх
140. else if (j % 2 != 0 && i > 0)
141. i--;
142. //или меняем столбец
143. else
144. j++;
145. counter++;
146. }
147. }
148. //Набор функций для схемы перестановки блоков
149. void Coursach::Lab3::copyFirstQuarter() {
150. for (int i = 0; i < arrayWidth / 2; i++)
151. for (int j = 0; j < arrayWidth / 2; j++)
152. \*(pArrTemp[i] + j) = \*(pArr[i] + j);
153. }
154. void Coursach::Lab3::pasteFirstQuarter() {
155. for (int i = 0; i < arrayWidth / 2; i++)
156. for (int j = 0; j < arrayWidth / 2; j++)
157. std::swap(\*(pArrTemp[i] + j), \*(pArr[i] + j + arrayWidth / 2));
158. }
159. void Coursach::Lab3::setElsLeftToRight(int minI) {
160. for (int i = minI; i < minI + arrayWidth / 2; i++)
161. for (int j = 0; j < arrayWidth / 2; j++)
162. std::swap(\*(pArr[i] + j), \*(pArr[i] + j + arrayWidth / 2));
163. }
164. void Coursach::Lab3::setElsUpToDown(int minJ) {
165. for (int i = 0; i < arrayWidth / 2; i++)
166. for (int j = minJ; j < minJ + arrayWidth / 2; j++)
167. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j));
168. }
169. void Coursach::Lab3::setElsDiagonalLeftToRight() {
170. for (int i = 0; i < arrayWidth / 2; i++)
171. for (int j = 0; j < arrayWidth / 2; j++)
172. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j + arrayWidth / 2));
173. }
174. void Coursach::Lab3::setElsDiagonalRightToLeft() {
175. for (int i = 0; i < arrayWidth / 2; i++)
176. for (int j = arrayWidth / 2; j < arrayWidth; j++)
177. std::swap(\*(pArr[i] + j), \*(pArr[i + arrayWidth / 2] + j - arrayWidth / 2));
178. }
179. void Coursach::Lab3::roundShift()
180. {
181. copyFirstQuarter();
182. setElsUpToDown(0);
183. setElsLeftToRight(arrayWidth / 2);
184. setElsUpToDown(arrayWidth / 2);
185. pasteFirstQuarter();
186. showArray(1);
187. secondMatrixPanel->Visible = true;
188. }
189. void Coursach::Lab3::diagonalShift()
190. {
191. setElsDiagonalLeftToRight();
192. setElsDiagonalRightToLeft();
193. showArray(1);
194. secondMatrixPanel->Visible = true;
195. }
196. void Coursach::Lab3::verticalShift()
197. {
198. setElsUpToDown(0);
199. setElsUpToDown(arrayWidth / 2);
200. showArray(1);
201. secondMatrixPanel->Visible = true;
202. }
203. void Coursach::Lab3::horizontalShift()
204. {
205. setElsLeftToRight(0);
206. setElsLeftToRight(arrayWidth / 2);
207. showArray(1);
208. secondMatrixPanel->Visible = true;
209. }
210. System::Void Coursach::Lab3::btnQuotersChange\_Click(System::Object^ sender, System::EventArgs^ e)
211. {
212. prepareMatrix();
213. switch (quotersOptionsCombo->SelectedIndex) {
214. case 0:
215. roundShift();
216. break;
217. case 1:
218. diagonalShift();
219. break;
220. case 2:
221. verticalShift();
222. break;
223. case 3:
224. horizontalShift();
225. break;
226. return;
227. }
228. }
229. float Coursach::Lab3::checkUserNum(String^ text)
230. {
231. float num;
232. if (System::Single::TryParse(System::Convert::ToString(text), num)) {
233. return num;
234. }
235. else {
236. showError("Введите число!");
237. errors++;
238. return -1;
239. }
240. }
241. void Coursach::Lab3::addition(float num){
242. for (int i = 0; i < arrayWidth; i++) {
243. for (int j = 0; j < arrayWidth; j++)
244. \*(pArr[i] + j) = \*(pArr[i] + j) + num;
245. }
246. }
247. void Coursach::Lab3::subtraction(float num)
248. {
249. for (int i = 0; i < arrayWidth; i++) {
250. for (int j = 0; j < arrayWidth; j++)
251. \*(pArr[i] + j) = \*(pArr[i] + j) - num;
252. }
253. }
254. void Coursach::Lab3::multiple(float num)
255. {
256. for (int i = 0; i < arrayWidth; i++) {
257. for (int j = 0; j < arrayWidth; j++)
258. \*(pArr[i] + j) = round(\*(pArr[i] + j) \* num \* 1000) / 1000;
259. }
260. }
261. void Coursach::Lab3::division(float num)
262. {
263. for (int i = 0; i < arrayWidth; i++) {
264. for (int j = 0; j < arrayWidth; j++)
265. \*(pArr[i] + j) = round(\*(pArr[i] + j)/num \* 1000)/1000;
266. }
267. }
268. System::Void Coursach::Lab3::btnAdd\_Click(System::Object^ sender, System::EventArgs^ e)
269. {
270. prepareMatrix();
271. float num = checkUserNum(textBoxNum->Text);
272. if (!errors) {
273. addition(num);
274. showArray(1);
275. secondMatrixPanel->Visible = true;
276. }
277. else return;
278. }
279. System::Void Coursach::Lab3::btnSub\_Click(System::Object^ sender, System::EventArgs^ e)
280. {
281. prepareMatrix();
282. float num = checkUserNum(textBoxNum->Text);
283. if (!errors) {
284. subtraction(num);
285. showArray(1);
286. secondMatrixPanel->Visible = true;
287. }
288. else return;
289. }
290. System::Void Coursach::Lab3::btnMtpl\_Click(System::Object^ sender, System::EventArgs^ e)
291. {
292. prepareMatrix();
293. float num = checkUserNum(textBoxNum->Text);
294. if (!errors) {
295. multiple(num);
296. showArray(1);
297. secondMatrixPanel->Visible = true;
298. }
299. else return;
300. }
301. System::Void Coursach::Lab3::btnDiv\_Click(System::Object^ sender, System::EventArgs^ e)
302. {
303. prepareMatrix();
304. float num = checkUserNum(textBoxNum->Text);
305. if (!errors) {
306. if (num != 0) {
307. division(num);
308. showArray(1);
309. secondMatrixPanel->Visible = true;
310. }
311. else showError("На 0 делить можно, но не здесь");
312. }
313. else return;
314. }
315. System::Void Coursach::Lab3::btnGetArray\_Click(System::Object^ sender, System::EventArgs^ e)
316. {
317. errors = 0;
318. hideError();
319. arrayWidth = getMatrixDegree();
320. fillMethod = getFillMethod();
321. if (errors == 0) {
322. if (workPanel->Visible == false) {
323. workPanel->Visible = true;
324. }
325. initialise();
326. matrixSetup();
327. selectScheme();
328. }
329. }
330. void Coursach::Lab3::insertSort()
331. {
332. int i, j, k, jp, ip;
333. int size = arrayWidth \* arrayWidth;
334. int flag;
335. do {
336. flag = 0;
337. for (k = 1; k < size; k++) {
338. //Вычисляем индексы текущего элемента
339. i = k / arrayWidth;
340. j = k - i \* arrayWidth;
341. //Вычисляем индексы предыдущего элемента
342. ip = (k - 1) / arrayWidth;
343. jp = (k - 1) - ip \* arrayWidth;
344. if (\*(pArr[i] + j) < \*(pArr[ip] + jp)) {
345. std::swap(\*(pArr[i] + j), \*(pArr[ip] + jp));
346. flag = 1;
347. }
348. }
349. } while (flag);
350. }
351. System::Void Coursach::Lab3::btnSort\_Click(System::Object^ sender, System::EventArgs^ e)
352. {
353. prepareMatrix();
354. insertSort();
355. showArray(1);
356. }

Приложение Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

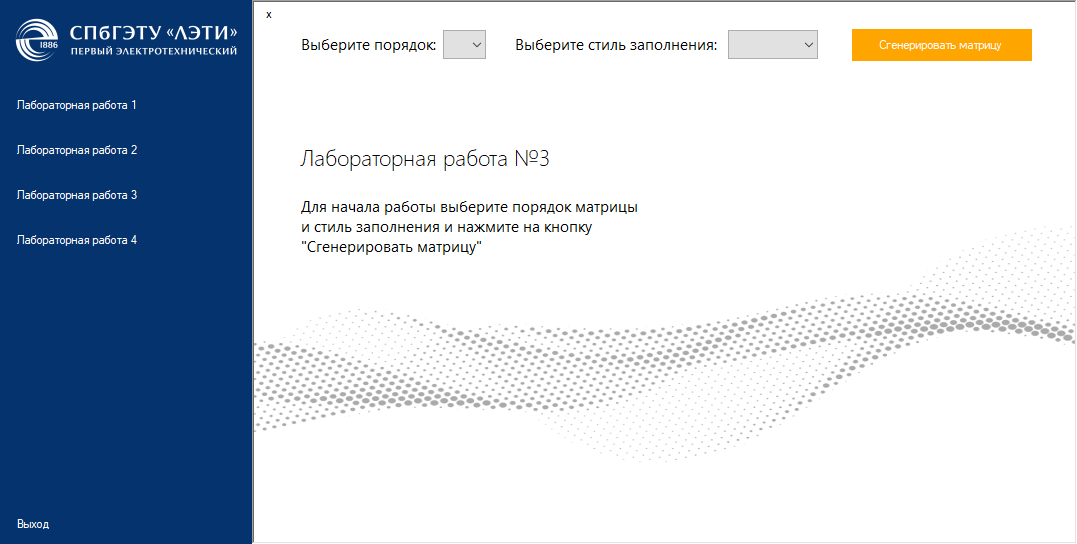


Рисунок 1 – Экран приветствия

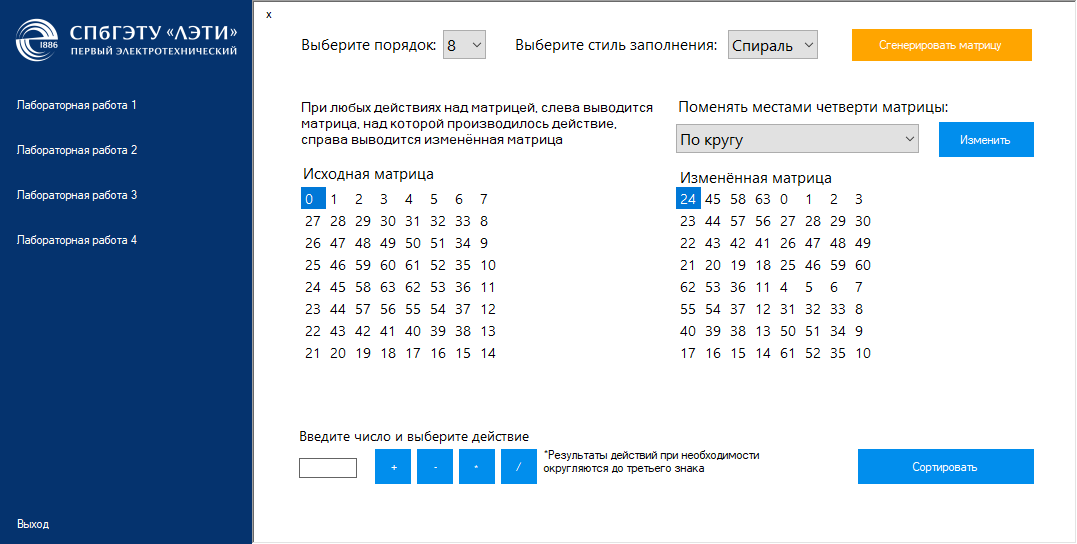


Рисунок 2 – Заполнение матрицы по спирали и замена четвертей матрицы по кругу

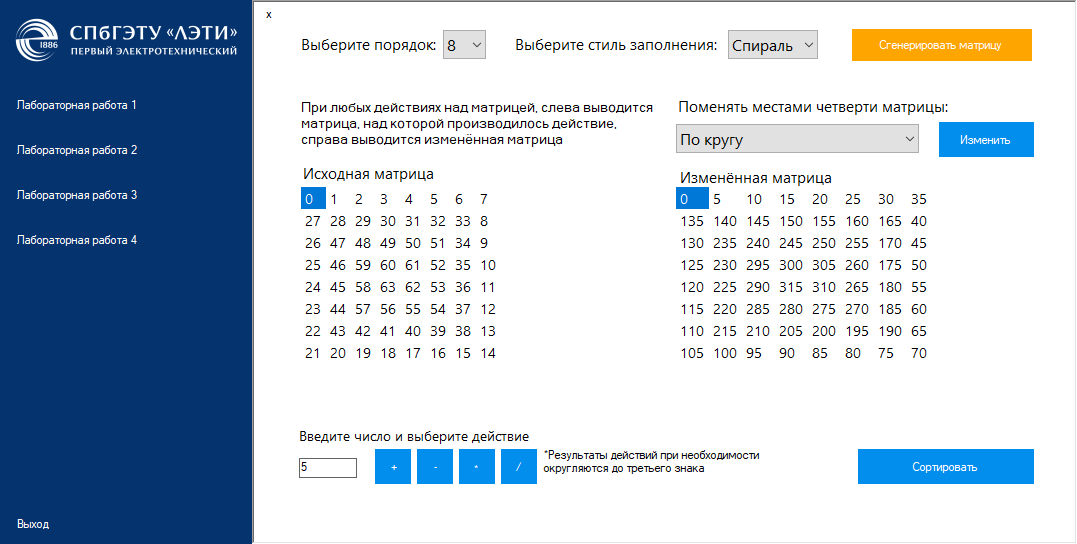


Рисунок 3 – Умножение матрицы на число

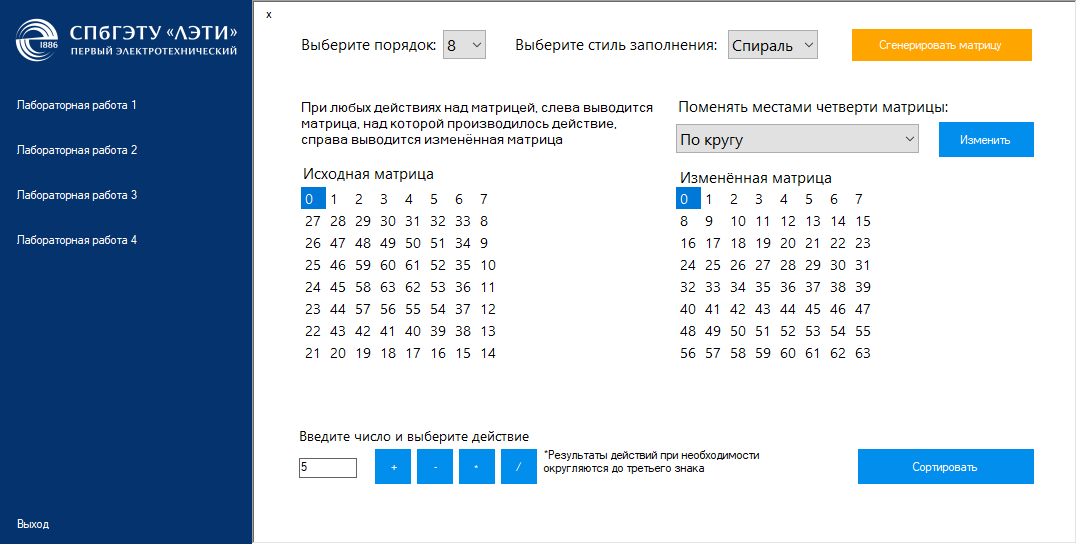


Рисунок 4 – Сортировка матрицы