**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)» (**МАИ)**

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 Вычислительные машины, системы и сети

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ **№ 1**

**“Формирование массивов**

**экспериментальных данных”**

Группа: *М30-209Б-19*

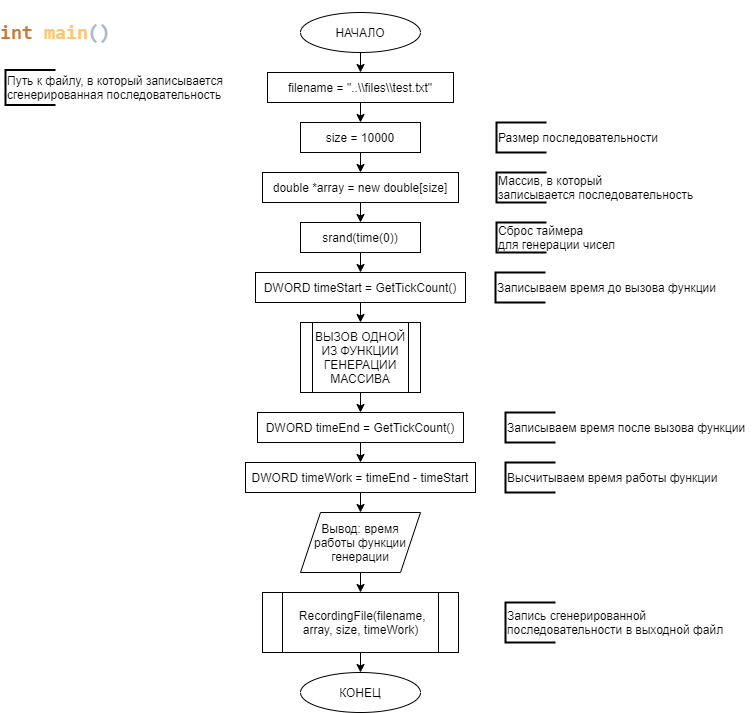
Выполнили:

*Кузнецов И.И*

Проверили:

**Москва 2020**

**Структурная схема алгоритма основной программы**

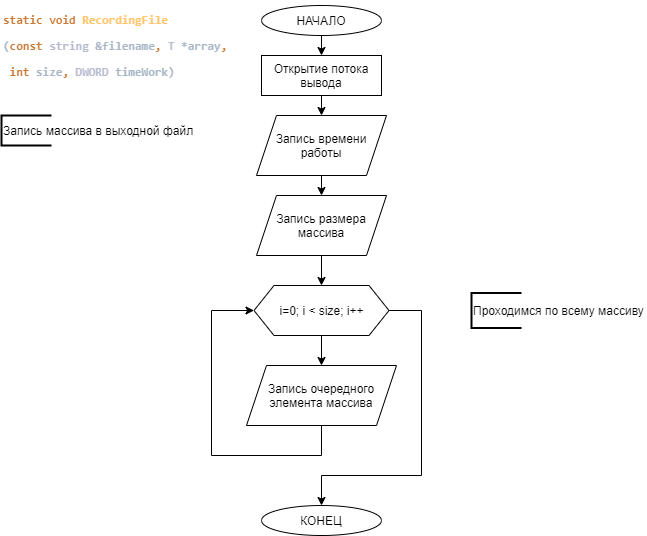


**Код основной программы**

int main() {  
 string filename\_1 = "..\\files\\test.txt";  
 setlocale(LC\_ALL, "Russian");  
 int size = 10000;  
 double \*array = new double[size];  
 srand(time(0));  
 DWORD timeStart = GetTickCount();  
 JustRandom(array, size); // функция для генерации последовательности  
 DWORD timeEnd = GetTickCount();  
 DWORD timeWork = timeEnd - timeStart;  
 cout << "Время: " << timeWork << " миллисекунд";  
 RecordingFile(filename\_1, array, size, timeWork);  
 return 0;  
}

**Описание функции для записи последовательности в файл**

**Структурная схема алгоритма**

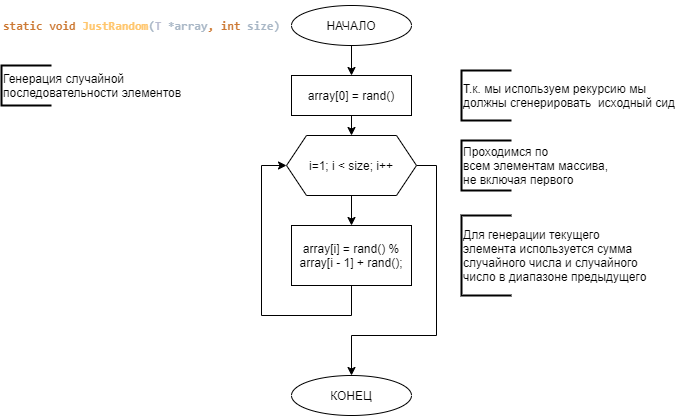


**Код алгоритма**

static void RecordingFile(const string &filename, T \*array, int size, DWORD timeWork) {  
 ofstream fileout(filename);  
 fileout << "Operating time (ms): " << timeWork << endl;  
 fileout << "Array size: " << size << endl;  
 for (int i = 0; i < size; ++i) {  
 fileout << array[i] << endl;  
 }  
 fileout.close();  
}

**Описание функции для генерации случайной последовательности**

**Структурная схема алгоритма**

****

**Код алгоритма**

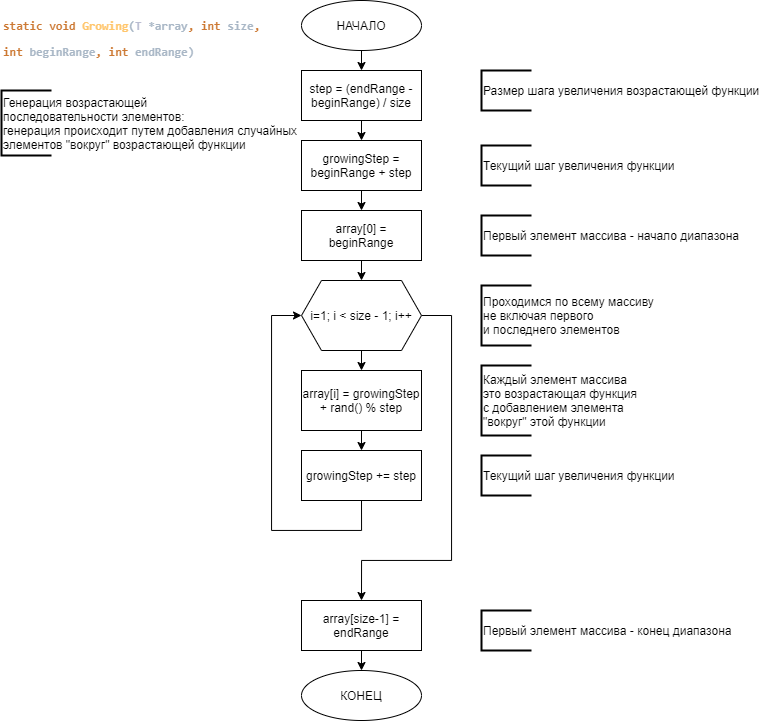
static void JustRandom(T \*array, int size) {  
 array[0] = rand();  
 for (int i = 1; i < size; ++i) {  
 array[i] = rand() % (int) array[i - 1] + rand();  
 }  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации возрастающей последовательности**

**Структурная схема алгоритма**

****

**Код алгоритма**

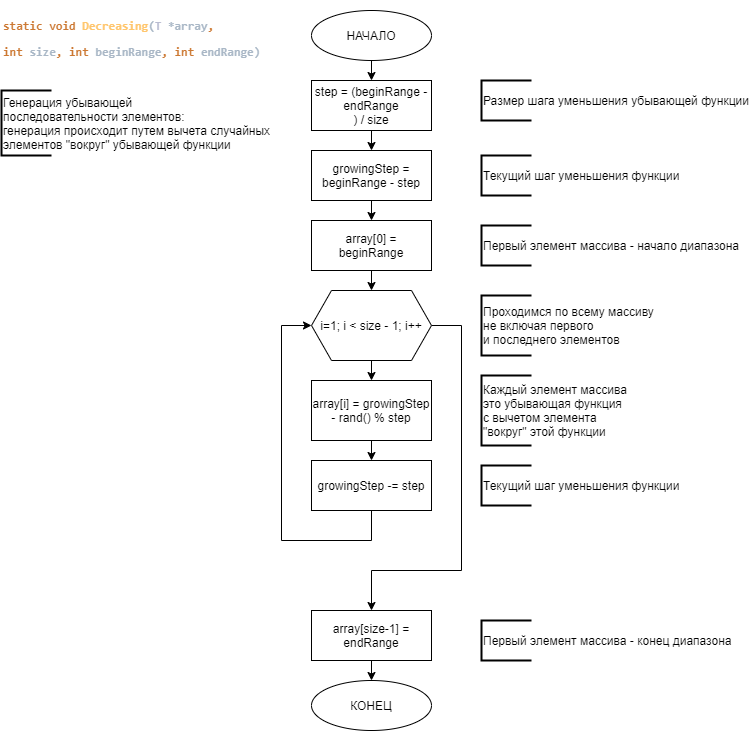
static void Growing(T \*array, int size, int beginRange, int endRange) {  
 T step = (endRange - beginRange) / size;  
 int growingStep = beginRange + step;  
 array[0] = beginRange;  
 for (int i = 1; i < size - 1; ++i) {  
 array[i] = growingStep + rand() % (int) step;  
 growingStep += step;  
 }  
 array[size - 1] = endRange;  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации убывающей последовательности**

**Структурная схема алгоритма**

****

**Код алгоритма**

static void Decreasing(T \*array, int size, int beginRange, int endRange) {  
 T step = (beginRange - endRange) / size;  
 int decreasingStep = beginRange - step;  
 array[0] = beginRange;  
 for (int i = 1; i < size - 1; ++i) {  
 array[i] = decreasingStep - rand() % (int) step;  
 decreasingStep -= step;  
 }

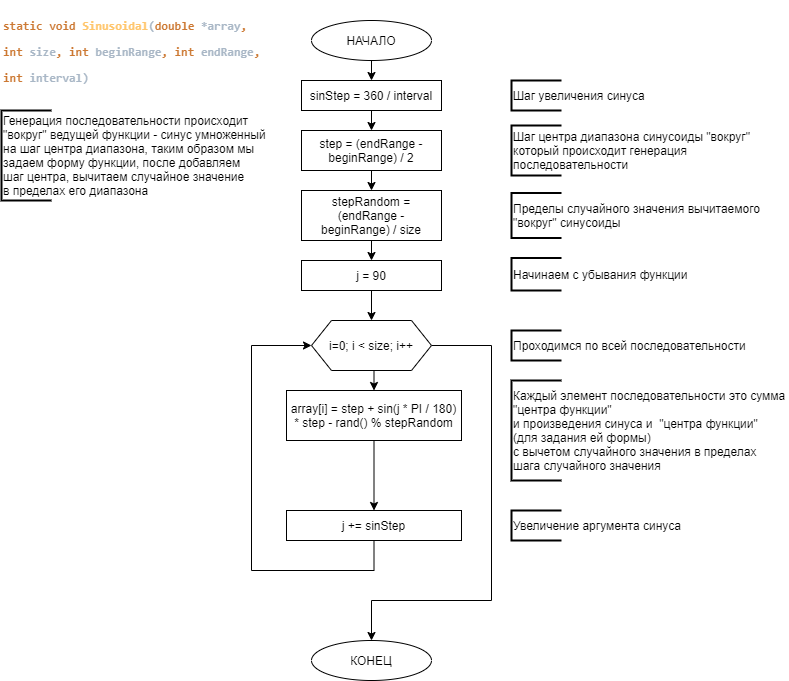
array [size - 1] = endRange;  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации синусоидальной последовательности**

**Структурная схема алгоритма**

****

**Код алгоритма**

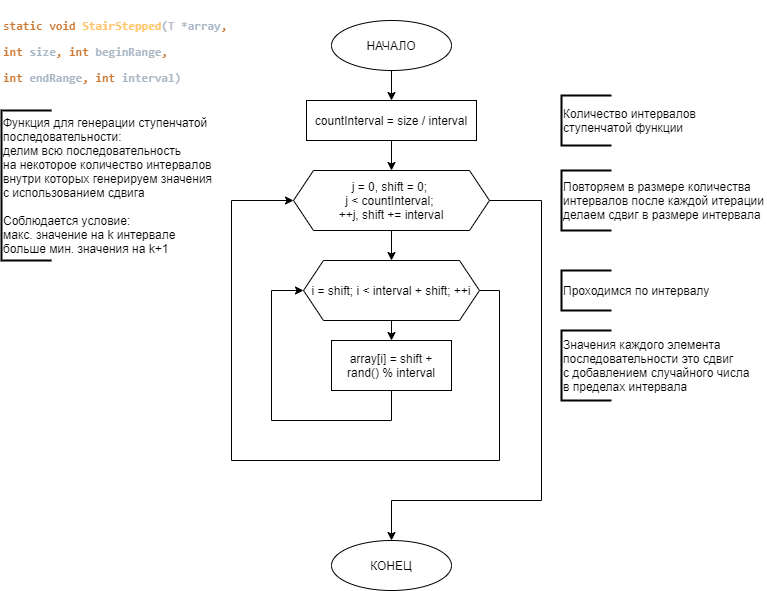
static void Sinusoidal(double \*array, int size, int beginRange, int endRange, int interval) {  
 double sinStep = 360.0 / interval;  
 int step = (endRange - beginRange) / 2;  
 int stepRandom = (endRange - beginRange) / size;  
 double j = 90.0;  
  
 for (int i = 0; i < size; ++i) {  
 array[i] = step + sin(j \* PI / 180) \* step - rand() % stepRandom;  
 j += sinStep;  
 }  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации ступенчатой последовательности**

**Структурная схема алгоритма**



**Код алгоритма**

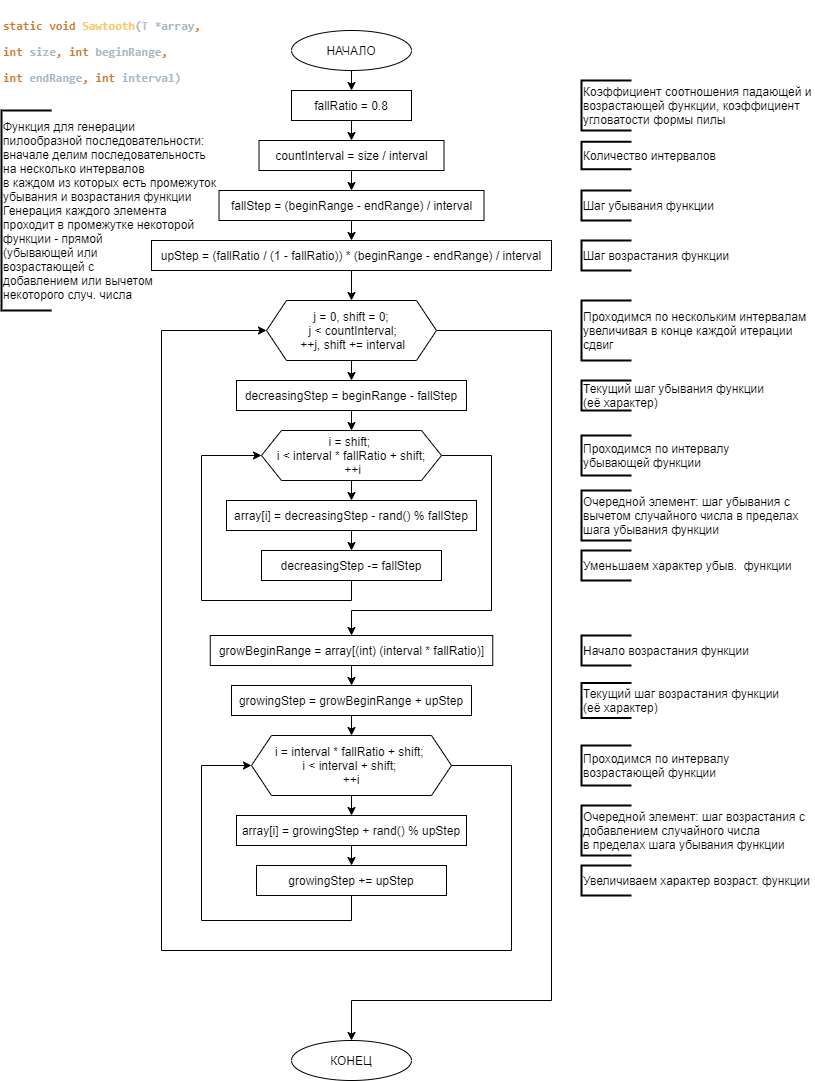
static void StairStepped(T \*array, int size, int beginRange, int endRange, int interval) {  
 int countInterval = size / interval;  
 for (int j = 0, shift = 0; j < countInterval; ++j, shift += interval) {  
 for (int i = shift; i < interval + shift; ++i) {  
 array[i] = shift + rand() % interval;  
 }  
 }  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации пилообразной последовательности**

**Структурная схема алгоритма**



**Код алгоритма**

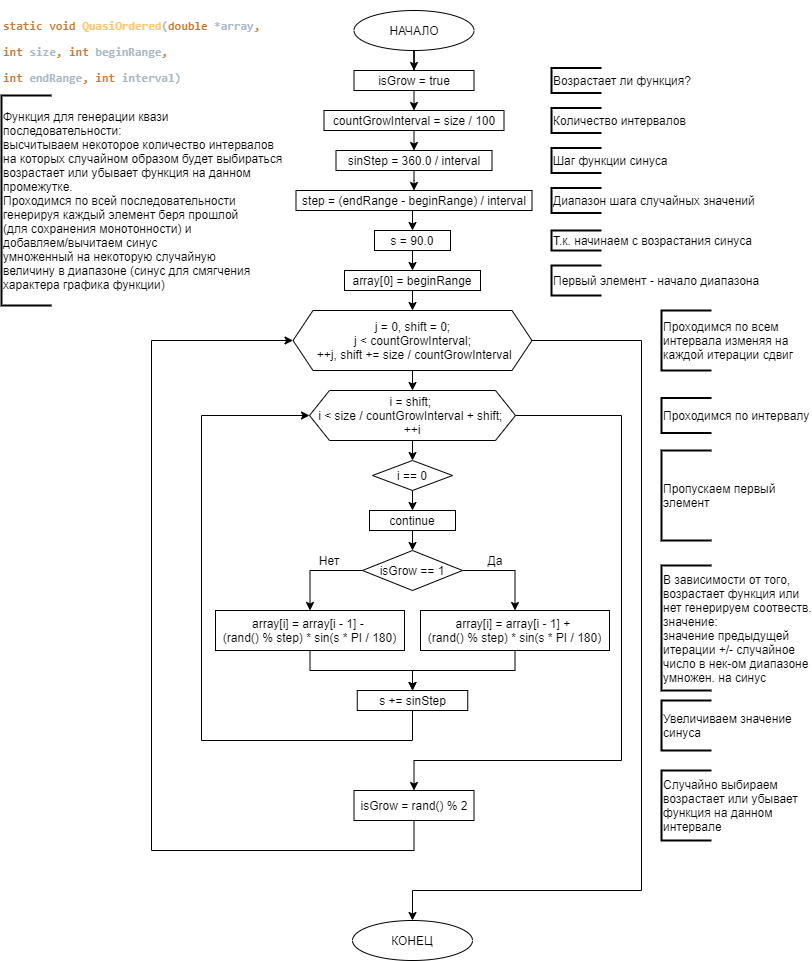
static void Sawtooth(T \*array, int size, int beginRange, int endRange, int interval) {  
 double fallRatio = 0.8;  
 int countInterval = size / interval;  
  
 int fallStep = (beginRange - endRange) / interval;  
 int upStep = (fallRatio / (1 - fallRatio)) \* (beginRange - endRange) / interval;  
  
 for (int j = 1, shift = 0; j <= countInterval; ++j, shift += interval) {  
 int decreasingStep = beginRange - fallStep;  
  
 for (int i = shift; i < interval \* fallRatio + shift; ++i) {  
 array[i] = decreasingStep - rand() % fallStep;  
 decreasingStep -= fallStep;  
 }  
  
 int growBeginRange = array[(int) (interval \* fallRatio)];  
 int growingStep = growBeginRange + upStep;  
  
 for (int i = interval \* fallRatio + shift; i < interval + shift; ++i) {  
 array[i] = growingStep + rand() % upStep;  
 growingStep += upStep;  
 }  
 }  
}

**График времени формирования последовательности**

**График последовательности**

**Описание функции для генерации квази-последовательности**

**Структурная схема алгоритма**



**Код алгоритма**

static void QuasiOrdered(double \*array, int size, int beginRange, int endRange, int interval) {  
 bool isGrow = true;  
 int countGrowInterval = size / 100;  
 double sinStep = 360.0 / interval;  
 int step = (endRange - beginRange) / interval;  
 double s = 90.0;  
 array[0] = beginRange;  
 for (int j = 1, shift = 0; j <= countGrowInterval; ++j, shift += size / countGrowInterval) {  
 for (int i = shift; i < size / countGrowInterval + shift; ++i) {  
 if (i == 0) continue;  
 if (isGrow == 1) {  
 array[i] = array[i - 1] + (rand() % step) \* sin(s \* PI / 180);  
 } else {  
 array[i] = array[i - 1] - (rand() % step) \* sin(s \* PI / 180);  
 }  
 s += sinStep;  
 }  
 isGrow = rand() % 2;  
 }  
}

**График времени формирования последовательности**

**График последовательности**