Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ СРЕДСТВО**

**«ВИЗУАЛИЗАЦИЯ СТАТИСТИЧЕСКИХ ДАННЫХ»**

БГУИР КП 1-40 01 01 023 ПЗ

Студент Прокопчук Д.М.

Руководитель Деменковец Д.В.

Минск 2023

СОДЕРЖАНИЕ

[Введение 5](#_Toc152614344)

[1 Анализ предметной области 6](#_Toc152614345)

[1.1 Обзор аналогов 6](#_Toc152614346)

[1.2 Постановка задачи 7](#_Toc152614347)

[2 Проектирование и разработка программного средства 8](#_Toc152614348)

[2.1 Структура программы 8](#_Toc152614349)

[2.2 Проектирование интерфейса программного средства 8](#_Toc152614350)

[2.3 Проектирование функционала программного средства 8](#_Toc152614351)

[2.4 Разработка программного средства 12](#_Toc152614352)

[3 Тестирование программного средства 19](#_Toc152614353)

[4 Руководство пользователя 21](#_Toc152614354)

[4.1 Интерфейс программного средства 21](#_Toc152614355)

[4.2 Управление программным средством 21](#_Toc152614356)

[Заключение 22](#_Toc152614357)

[Список использованных источников 23](#_Toc152614358)

[Приложение А (информационное) Исходный код программы 24](#_Toc152614359)

# ВВЕДЕНИЕ

Визуализация данных — это представление данных в виде, который обеспечивает наиболее эффективную работу человека по их изучению. Визуализация данных находит широкое применение в научных и статистических исследованиях (в частности, в прогнозировании, интеллектуальном анализе данных, бизнес-анализе), в педагогическом дизайне для обучения и тестирования, в новостных сводках и аналитических обзорах. Визуализация данных связана с визуализацией информации, инфографикой, визуализацией научных данных, разведочным анализом данных и статистической графикой. Визуализация данных позволяет наглядно представить сложные статистические показатели, раскрыть скрытые закономерности и тренды, а также облегчить восприятие и анализ информации.

Истоки представления данных в виде таблиц, диаграмм и карт прослеживаются с древнейших времён. Ощутимая потребность в качественном представлении информации стала возникать в эпоху Возрождения, с появлением больших количеств данных и визуальной информации из географии, астрономии, геометрии, статистики и других наук. В первой половине XIX века наблюдался значительный рост работ, в которых использовалось графическое отображение данных. К середине века были изобретены все основные типы представления данных: столбчатые и круговые диаграммы, гистограммы, линейные графики, графики временных рядов, контурные диаграммы и т. д. Тенденция роста пошла на спад в начале XX века, уступив место точной математике. Тем не менее, именно в этот период стали появляться учебники и курсы по графическим методам представления данных, а сами графики стали использоваться не только для представления результатов, но и для исследования информации и выдвижения гипотез в астрономии, физике, биологии и других науках.

Целью данной курсовой работы является разработка программного решения на языке программирования C++ с использованием WinAPI, которое позволит визуализировать статистические данные.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Программное средство “Визуализация статистических данных” представляет собой реализующее визуализацию текстовых или числовых данных в виде графиков.

## Обзор аналогов

1.1.1 “Datawrapper”

“Datawrapper” — это онлайн-инструмент для визуализации данных, который позволяет пользователям создавать интерактивные и адаптивные диаграммы, карты и другие визуализации. Он предоставляет удобный интерфейс, который позволяет физическим лицам, журналистам и организациям ясно и привлекательно представлять данные, даже без обширных навыков программирования или дизайна. С помощью Datawrapper вы можете импортировать данные из различных источников, таких как электронные таблицы или базы данных, а затем настраивать и визуализировать данные с помощью различных типов диаграмм, включая линейные графики, столбчатые диаграммы, круговые диаграммы, точечные графики и многое другое. Платформа предлагает широкий спектр настроек для индивидуальной настройки, позволяя пользователям изменять цвета, метки, оси и другие визуальные элементы в соответствии с их желаемым стилем и презентацией. На рисунке 1.1 предоставлен внешний вид онлайн-инструмента “Datawrapper”.

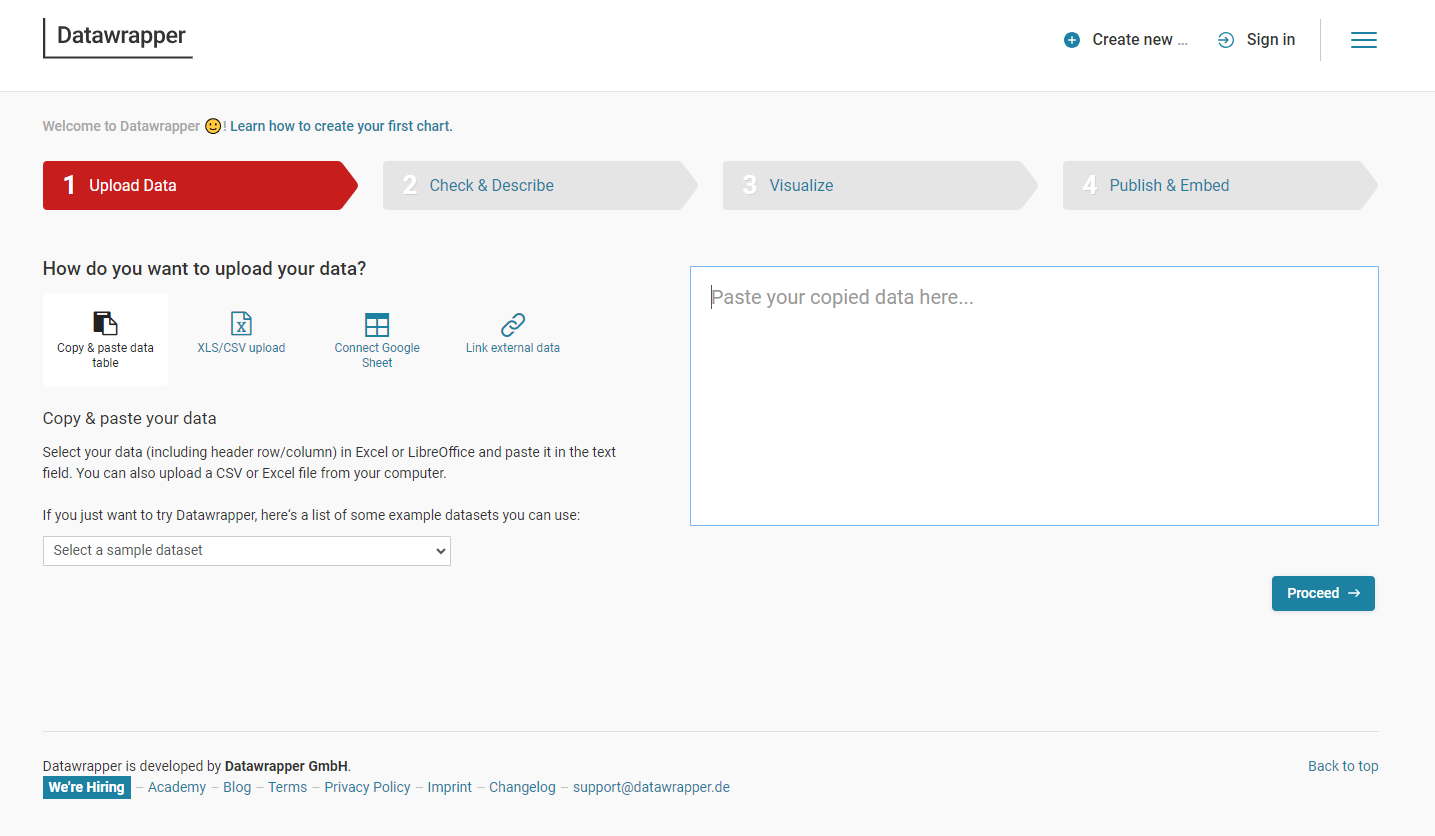


Рисунок 1.1 – Интерфейс онлайн-инструмента “Datawrapper”

Datawrapper также предлагает функции для подписей данных, аннотаций и интерактивности. Пользователи могут добавлять всплывающие подсказки, легенды и аннотации, чтобы предоставить дополнительный контекст и информацию к своим визуализациям. Созданные диаграммы могут быть встроены на веб-сайты, распространены через социальные сети или загружены в форматах PNG или SVG для использования в печатных материалах или других приложениях.

## Постановка задачи

В рамках данной курсовой работы планируется разработать программное средство генерации и отображения информации на экране. В процессе разработки должны быть реализованы следующие функции:

* генерация информации;
* чтение данных из файла с автоматическим определением типа;
* отображение числовых данных;
* отображение строковых данных.

Для разработки программного средства будет использоваться язык программирования С++ и среда разработки Microsoft Visual Studio 2022.

# ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано 6 модулей:

* Main – главный модуль, в котором обрабатываются события окна;
* WinApi – модуль, предоставляющий интерфейс взаимодействия с функциями генерации данных и чтения данных, а также визуализации;
* Drawer – модуль, отвечающий за рисование графиков;
* Logic – модуль, отвечающий за расчеты внутри программы.

## 2.2 Проектирование интерфейса программного средства

2.2.1 Главное окно

Главное окно приложения содержит весь доступный функционал: кнопку генерации данных, кнопку чтения данных, кнопку визуализации данных и пространство для вывода графиков. Макет главного окна приложения представлен на рисунке 2.1.

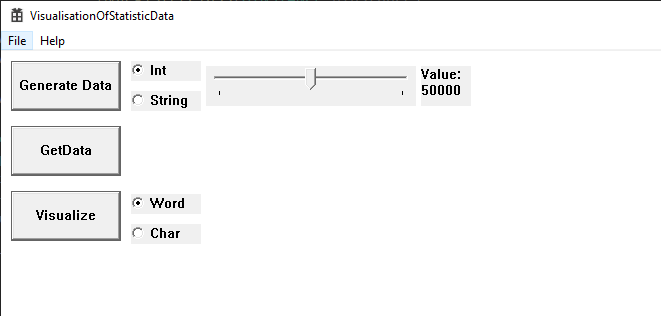


Рисунок 2.1 – Главное окно приложения

## 2.3 Проектирование функционала программного средства

В драйвере должны быть реализованы следующие методы:

* генерация информации;
* чтение данных из файла с автоматическим определением типа;
* отображение данных;
* подсчет частоты.

2.3.1 Генерация информации

Генерация информации позволяет пользователю не искать данные в различных форматах, а просто создать случайные данные и отобразить их. При генерации будет использоваться основная функция generateData. Блок-схема алгоритма данной функции приведена на рисунке 2.2.



Рисунок 2.2 – Блок-схема функции генерации данных.

2.3.2 Чтение данных из файла с автоматическим определением типа

Чтение данных из файла позволяеет пользователю как прочитать позаимствованные данные, так и те, которые пользователь сгенерировал. При чтении будет использоваться основная функция getData. Блок-схема алгоритма данной функции приведена на рисунке 2.3.



Рисунок 2.3 – Блок-схема функции чтения данных.

2.3.3 Отображение данных

Отображение данных позволяет пользователю просмотреть, какие числа, буквы или слова чаще встречаются в наборе в графическом виде. Данная процедура получает набор данных, через модуль логики считает частоту встречаемости и выводит на экран. Блок-схема алгоритма процедуры DrawPlot приведена на рисунке 2.4.



Рисунок 2.4 – Блок-схема процедуры отображения данных

2.3.4 Подсчет частоты

Для отображения данных в виде графика необходимо каким-то образом сгруппировать данные. В данном программном средстве группировка данных будет происходить по частоте встречаемости. Функция подсчета будет получать вектор данных, проходить по нему и возвращать словарь, где ключи – это данные, а значения – это частота их встречаемости. Блок-схема алгоритма процедуры countFreq приведена на рисунке 2.5.



Рисунок 2.5 – Блок-схема функции подсчета частоты

## 2.4 Разработка программного средства

2.4.1 Генерация информации

При генерации информации используется механизм генерации случаных чисел предоставленный в C++ заголовком <random>. Код метода генерации информации представлен ниже.

DataGenerator::Data DataGenerator::generateData() {

Data output;

// определение типа данных

output.type = DataGenerator::generateDataType;

//создание генератора псевдослучайных чисел

std::random\_device rd;

std::mt19937 gen(rd());

if (DataGenerator::generateDataType == Types::Type::INT) {

output.charData = new char\* [DataGenerator::sizeOfData];

if (DataGenerator::isBounds) {

// определение размера генерируемых чисел

std::uniform\_int\_distribution<int> dis(DataGenerator::minInt, DataGenerator::maxInt);

for (int i = 0; i < DataGenerator::sizeOfData; i++) {

int randNum = dis(gen);

std::string strNum = std::to\_string(randNum);

output.charData[i] = new char[strNum.length() + 1];

strcpy\_s(output.charData[i], strNum.length() + 1, strNum.c\_str());

}

}

else {

// определение размера генерируемых чисел

std::uniform\_int\_distribution<int> dis(DataGenerator::fixedMin, DataGenerator::fixedMax);

for (int i = 0; i < DataGenerator::sizeOfData; i++) {

int randNum = dis(gen);

char\* temp = new char[2];

temp[0] = '0' + randNum;

temp[1] = '\0';

output.charData[i] = temp;

}

}

}

else {

// определение длинн генерируемых случайных строк

std::uniform\_int\_distribution<int> lengthDistribution(2, maxStringLength);

// определение генерируемого набора символов

std::uniform\_int\_distribution<int> charDistribution(0, charsetSize - 1);

output.charData = new char\* [DataGenerator::sizeOfData];

for (int i = 0; i < DataGenerator::sizeOfData; i++) {

int length = lengthDistribution(gen);

char\* temp = new char[length + 1];

for (int j = 0; j < length; j++) {

temp[j] = charset[charDistribution(gen)];

}

temp[length] = '\0';

output.charData[i] = temp;

}

}

// возврат структуры, содержащей сгенерированные данные и их тип

return output;

}

Листинг 1 – Генерация информации

2.4.2 Чтение данных из файла с автоматическим определением типа

При чтении данных будет использовать метод ReadFile, а также возможности C++ для разбивания цельной строки на элементы вектора по разделителю пробелу. Код метода чтения информации и связанных с ним представлен ниже.

DataCatcher::Data DataCatcher::GetData(const wchar\_t\* fileName) {

DataCatcher::Data catchedData;

try {

FileReader\* reader = new FileReader(fileName);

// чтение из файла

std::vector<char\*> buffer = reader->readAllToChar();

// определение типа данных

catchedData.type = DataCatcher::ResolveDataType(buffer[0]);

catchedData.charData = buffer;

// удаление ссылки на объект

delete reader;

} catch (std::exception e) {

MessageBox(NULL, L"Error while reading file", L"Error", MB\_ICONERROR);

}

return catchedData;

}

std::vector<char\*> FileReader::readAllToChar() {

// чтение всего файла в вектор char

std::vector<char> buffer = FileReader::readAll();

std::vector<char\*> elements;

std::istringstream iss(buffer.data());

std::string element;

// разделение данных по пробелам

while (std::getline(iss, element, ' '))

{

char\* elementPtr = new char[element.size() + 1];

strcpy\_s(elementPtr, element.size() + 1, element.c\_str());

// добавление разделенных элементов в новый вектор

elements.push\_back(elementPtr);

}

return elements;

}

std::vector<char> FileReader::readAll() {

if (isOpened) {

DWORD fileSize;

// определение размера файла

fileSize = GetFileSize(FileReader::file, NULL);

DWORD bytesRead;

std::vector<char> buffer(fileSize + 1);

// чтение файла в buffer

if (!ReadFile(FileReader::file, buffer.data(), fileSize, &bytesRead, NULL))

{

// обработка ошибки чтеения

CloseHandle(FileReader::file);

throw std::exception("Error while reading file");

}

return buffer;

} else {

std::vector<char> vect;

return vect;

}

}

Листинг 2 – Чтение информации из файла

2.4.3 Отображение данных

При отображении данных сперва подсчитаеется частота встречаемости значений в общих данных, затем будут выведены столбцы, отображающие частоту, а затем нарисованы оси графика. Код метода отображения данных и связанных с ним представлен ниже.

void Drawer::drawPlot(std::vector<char\*> data, HDC hdc, HWND hWnd) {

// установление режима фона на TRANSPARENT

SetBkMode(hdc, TRANSPARENT);

std::map<std::string, double> KSTest;

// проведение расчетов для целочисленных данных

if (Drawer::typeOfData == Types::Type::INT) {

KSTest = StatisticResolver::KSTestForInt(data);

}

// подсчет частоты встречаемости

std::optional<std::map<std::string, int>> result = Drawer::countFreq(data);

if (!result) {

// обработка ошибок

return;

}

std::map<std::string, int> frequency = result.value();

// отображение столбцов

Drawer::drawBars(hdc, hWnd, frequency);

// отображение осей

Drawer::drawAxes(hdc, hWnd, frequency);

}

void Drawer::drawBars(HDC hdc, HWND hwnd, std::map<std::string, int> frequency) {

// подсчет самого часто встречаемого элемента

auto maxElement = std::max\_element(frequency.begin(), frequency.end(),

[](const auto& pair1, const auto& pair2) {

return pair1.second < pair2.second;

});

// определение ширины столбца

int barWidth = Drawer::chartWidth / frequency.size();

int i = 0;

for (const auto& pair : frequency)

{

// определение высоты столбца

int barHeight = (pair.second \* Drawer::chartHeight) / maxElement->second;

RECT barRect, textRect;

// определение позиции столбца на экране

barRect.left = Drawer::chartX + i \* barWidth;

barRect.top = Drawer::chartY + Drawer::chartHeight - barHeight;

barRect.right = barRect.left + barWidth;

barRect.bottom = Drawer::chartY + chartHeight;

// определение позиции текста содержащего значения

textRect.left = barRect.left;

textRect.right = barRect.right;

textRect.bottom = barRect.top;

textRect.top = textRect.bottom - 30;

std::wstring label = std::to\_wstring(pair.second);

// создание кисти разного цвета

HBRUSH brush = CreateSolidBrush(RGB(255 - i \* 20, i \* 30, i \* 40));

// рисование столбца

FillRect(hdc, &barRect, brush);

// вывод текста

DrawText(hdc, label.c\_str(), -1, &textRect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

DeleteObject(brush);

i++;

}

}

void Drawer::drawAxes(HDC hdc, HWND hwnd, std::map<std::string, int> frequency) {

int barWidth = Drawer::chartWidth / frequency.size();

// рисование 2 линий как 2 осей

MoveToEx(hdc, Drawer::chartX, Drawer::chartY + Drawer::chartHeight, NULL);

LineTo(hdc, Drawer::chartX + Drawer::chartWidth, Drawer::chartY + Drawer::chartHeight);

MoveToEx(hdc, Drawer::chartX, Drawer::chartY, NULL);

LineTo(hdc, Drawer::chartX, Drawer::chartY + Drawer::chartHeight);

int i = 0;

// вывод подписей к осям

for (const auto& pair : frequency)

{

std::wstring label(pair.first.begin(), pair.first.end());

RECT labelRect;

labelRect.left = Drawer::chartX + i \* barWidth;

labelRect.top = Drawer::chartY + Drawer::chartHeight + 2;

labelRect.right = labelRect.left + barWidth;

labelRect.bottom = labelRect.top + 20;

DrawText(hdc, label.c\_str(), -1, &labelRect, DT\_CENTER | DT\_VCENTER);

i++;

}

}

Листинг 3 – Отображение данных

2.4.4 Подсчет частоты

При подсчете частоты существует 2 возможных метода: подсчет букв и подсчет слов, для строковых данных можно применить оба метода, для числовых только 1. Код метода подсчета частоты и связанных с ним представлен ниже.

std::optional<std::map<std::string, int>> Drawer::countFreq(std::vector<char\*> data) {

// создание ссылки на объект, считающий частоту

FrequencyCounter\* counter = new FrequencyCounter();

std::map<std::string, int> frequency;

try {

// подсчет частоты букв

if (Drawer::type == Types::VisualizationType::CHAR) {

frequency = counter->countAlphabetFrequency(data);

}

// подсчет частоты слов/чисел

else if (Drawer::type == Types::VisualizationType::WORD) {

frequency = counter->countWordsFrequency(data);

}

return frequency;

}

catch (std::exception e) {

MessageBox(NULL, L"Error when counting frequency", L"Error", MB\_ICONERROR);

return std::nullopt;

}

}

std::map<std::string, int> FrequencyCounter::countAlphabetFrequency(std::vector<char\*> data) {

std::map<std::string, int> freq;

for (auto line : data) {

char\* temp = line;

while (\*temp) {

// перевод букв в верхний регистр

char c = toupper(\*temp);

// проверка вхождения в алфавит

if (c >= 'A' && c <= 'Z') {

std::string stringVal(1, c);

// создание нового ключа в map или увеличение значения для существуюшего

freq[stringVal]++;

// обработка ошибки, когда ключей больше букв в алфавите

if (freq.size() > FrequencyCounter::maxItems) {

throw new std::exception("This data don't have something in common");

}

}

temp++;

}

}

return freq;

}

std::map<std::string, int> FrequencyCounter::countWordsFrequency(std::vector<char\*> data) {

std::map<std::string, int> freq;

// подсчет частоты слов

for (auto line : data) {

std::string str(line);

std::transform(str.begin(), str.end(), str.begin(),

[](unsigned char c) { return std::tolower(c); });

freq[str]++;

}

// создание multimap для сортировки значений по частоте

std::multimap<int, std::string> sortedFreq;

//вставка обратных пар для сортировки по количеству вхождений

for (auto& pair : freq) {

sortedFreq.insert({ pair.second, pair.first });

}

int count = 0;

std::map<std::string, int> resultMap;

// сохранение 30 самых часто встречаемых слов/чисеел в данных

for (auto itr = sortedFreq.rbegin(); itr != sortedFreq.rend() && count < FrequencyCounter::wordsDisplay; ++itr) {

resultMap[itr->second] = itr->first;

count++;

}

return resultMap;

}

Листинг 4 – Подсчет частоты

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование играет ключевую роль в процессе разработки, позволяя выявить и исправить ошибки, а также убедиться в соответствии функциональности требованиям и ожиданиям пользователей.

Таблица 1 – Тест-кейсы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Предусловия** | **Шаги** | **Ожидаемый результат** | **Реальный результат** |
| 1 | Генерация данных | 1. Выбран тип генерируемых данных(INT, STRING)  2. Выбрано количество генерируемых данных | 1. Нажать кнопку “Generate Data”  2. Выбрать файл для сохранения | Появится сообщение на экране об успешной генерации, а также о том, что данные записаны в файл | На экране появилось окно с текстом “Data successfully writed to file” |
| 2 | Чтение данных | 1. Выбран файл для чтения  2. В файле находятся какие-либо данные | 1. Нажать кнопку “GetData” | Появится сообщение на экране об успешном чтении данных из файла | На экране появилос окно с текстом “Data successfully read from file” |

Продолжение таблицы 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Предусловия** | **Шаги** | **Ожидаемый результат** | **Реальный результат** |
| 3 | Рисование графика для чисел на экране | 1. Выбран режим WORD  2. Данные загружены из файла | 1. Нажать кнопку “Visualize” | На экране появится столбчатая диаграмма, описывающая частоту встречаемости чисел | На экране появилась диаграмма описывающая 20 самых часто встречаемых чисел |
| 4 | Рисование графика для букв на экране | 1. Выбран режим Char  2. Данные загружены из файла | 1. Нажать кнопку “Visualize” | На экране появится столбчатая диаграмма, описывающая частоту букв | На экране появилась диаграмма, описывающая частоту встречаемости букв в данных |
| 5 | Генерация чисел в границах | 1.Выбран режим генерации INT  2.Выбраны границы генерации  (-100, 100)  3.Выбран файл для сохранения | 1. Нажать кнопку “Generate Data” | В файл запишутся числа в диапазоне от -100 до 100 включительно | В файл записаны числа в диапазоне от – 100 до 100 |

В результате тестирования программного средства, все тесты прошли успешно, подтверждая стабильность и соответствие функциональности поставленным требованиям.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 4.1 Интерфейс программного средства

4.1.1 Главное окно

Главное окно представляет собой набор кнопок для вызова основых функций приложения. Здесь содрежаться кнопки генерации данных, чтения данных, визаулизации, а также варианты выбора различных типов данных. Внешний вид главного окна приложения представлен на рисунке 5.1.

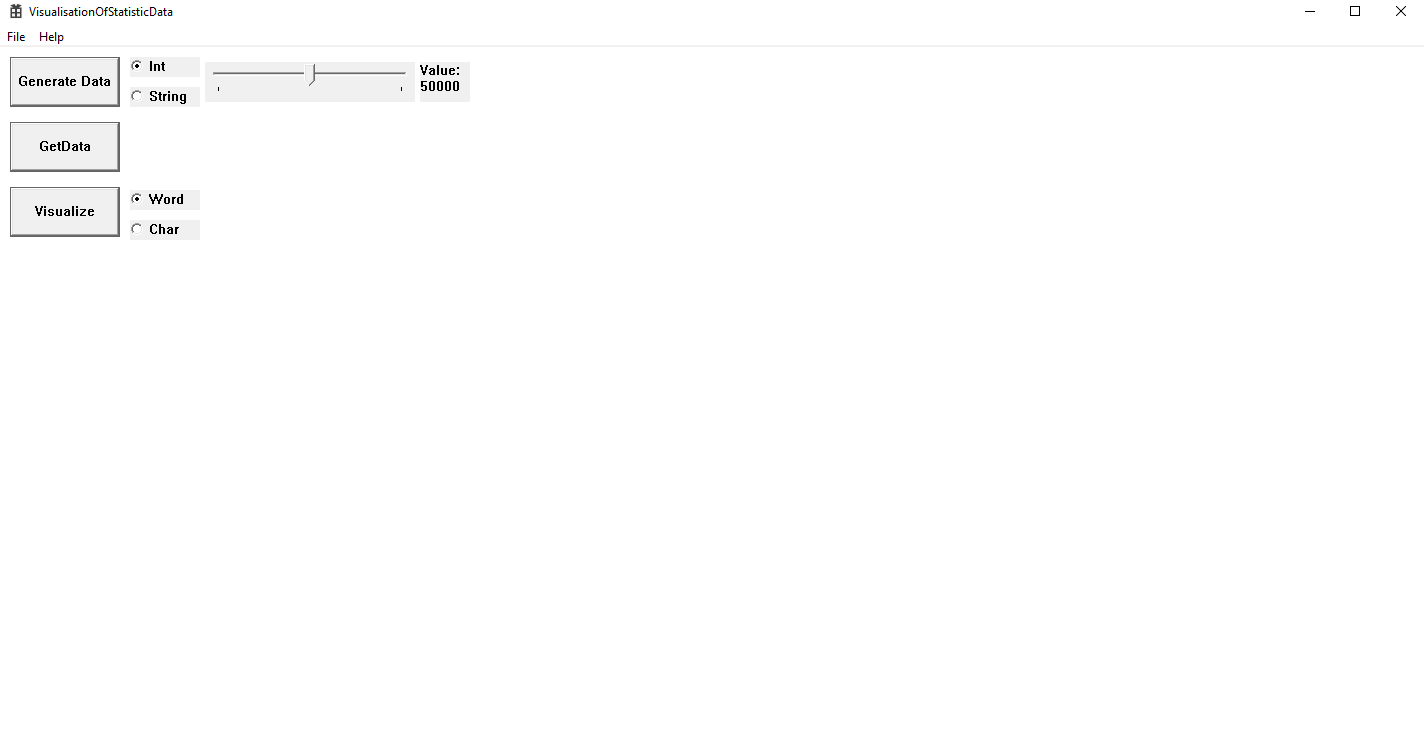


Рисунок 5.1 – Главное окно приложения

## 4.2 Управление программным средством

Для того, чтобы начать работу с программным средством необходимо запустить его. Далее чтобы опробовать функционал необходимо либо иметь собственный набор данных, разделенных пробелом, либо сгенерировать случайный используя кнопку “Generate Data”. Далее необходимо получить данные из файла, для этого нужно нажать кнопку “GetData” и выбрать файл, из которого будет произведено чтении данных. При получении успешного сообщения о чтении пользователь может выбрать тип отображения данных рядом с кнопкой “Visualize”, а именно “WORD” или “CHAR”. Для целых чисел доступно только “WORD”. После выбора желаемой опции следует нажать на кнопку “Visualize”, программа сгенерирует столбчатую диаграмму и выведет ее на экран. Также стоит упомянуть, что при генерации чисел пользователю предоставляется возможность выбрать диапазон генерируемых значений, а именно от -5000 до 1000001.

# ЗАКЛЮЧЕНИЕ

В заключении можно сказать, что было создано вспомогательное программное средство для визуализации данных. В рамках данного курсового проекта было разработано программное средство “Визуализация статистических данных”, которое позволит пользователям операционной системы Windows визуализировать некоторые данныые. При разработке данного средства были успешно выполнены следующие поставленные задачи:

* генерация информации;
* чтение данных из файла с автоматическим определением типа;
* отображение числовых данных;
* отображение строковых данных.

Для успешного выполнения поставленных задач потребовалось изучить

основные принципы работы с файлами с помощью WinAPI, а также изучить основные способы вывода графической информации на экран, также взаимодействие с диалоговыми окнами.

Существует множество способов улучшить данное программное средство: добавить больше графических отображений, улучшить графическую составляющую, добавить генерацию более-менее осмысленных текстов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Визуализация данных [Электронный ресурс]. ‒ Режим доступа:

https://ru.wikipedia.org/wiki/Визуализация\_данных.

[2] “Datawrapper” [Электронный ресурс]. ‒ Режим доступа: https://www.datawrapper.de/.

[3] Файлы в WinAPI [Электронный ресурс]. ‒ Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/.

[4] Инструмент рисования [Электронный ресурс]. ‒ Режим доступа:

https://learn.microsoft.com/en-us/windows/win32/api/\_gdi/.

[5] Visual Studio 2022 [Электронный ресурс]. ‒ Режим доступа:

<https://visualstudio.microsoft.com/ru/>.

[6] СТП 01–2017, Стандарт предприятия, дипломные проекты, общие требования [Электронный ресурс]. – Электронные данные. – Режим доступа: https://library.bsuir.by/m/12\_101945\_1\_141950.pdf.

[7] Манцнер, Т. Визуализация данных. Полный и исчерпывающий курс для начинающих / М.А. Райтман. – Бомбора, 2023. – 464 c.

[8] Бизянов, Е.Е. Системное программирование / Е.Е. Бизянов. – Инфра-Инженерия, 2023. – 368 c.

[9] Литвиненко, Н.А. Технология программирования на С++ Win32 API-приложения / Литвиненко, Н.А. – СПб.: БХВ, 2010. – 281 c.

# ПРИЛОЖЕНИЕ А. Исходный код программы

#include "framework.h"

#include "VisualisationOfStatisticData.h"

#include "DataGenerator.h"

#include "commdlg.h"

#include "windows.h"

#include <commctrl.h>

#include "Constants.h"

#include "Drawer.h"

#include "Types.h"

#include "WinApi.h"

//Buttons

HWND generateButton;

HWND hookButton;

HWND visualizeButton;

//checkBoxes

HWND strTypeCheckBox;

HWND intTypeCheckBox;

HWND charStyleCheckBox, wordStyleCheckBox;

//trackBar

HWND amountTrackBar;

//labels

HWND trackBarLabel;

//edits

HWND hLowerBoundEdit, hUpperBoundEdit;

HINSTANCE hInst;

WCHAR szTitle[MAX\_LOADSTRING];

WCHAR szWindowClass[MAX\_LOADSTRING];

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

void GenerateData(HWND hWnd);

INT\_PTR CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam);

std::vector<char\*> data;

Types::Type inputType;

bool isPaint = false;

std::pair<int, int> bounds;

bool isBounds = false;

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_VISUALISATIONOFSTATISTICDATA, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_VISUALISATIONOFSTATISTICDATA));

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_HSCROLL:

{

int x = 0;

if (lParam == reinterpret\_cast<LPARAM>(amountTrackBar))

{

WinApi::getInstance().UpdateLabelValue(trackBarLabel, amountTrackBar);

}

break;

}

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case GENERATE\_DATA:

{

Types::Type type = IsDlgButtonChecked(hWnd, INT\_TYPE) ? Types::Type::INT : Types::Type::STRING;

if (type == Types::Type::INT) {

DialogBox(hInst, MAKEINTRESOURCE(IDD\_DIALOG\_BOUNDS), hWnd, DialogProc);

}

GenerateData(hWnd);

isBounds = false;

break;

}

case HOOK\_DATA:

{

DataCatcher::Data2 info = WinApi::getInstance().HookData(hWnd, nullptr);

data = info.charData;

inputType = info.type;

if (inputType == Types::Type::INT) {

if (IsDlgButtonChecked(hWnd, CHAR\_TYPE)) {

SendMessage(charStyleCheckBox, BM\_SETCHECK, BST\_UNCHECKED, 0);

SendMessage(wordStyleCheckBox, BM\_SETCHECK, BST\_CHECKED, 0);

}

EnableWindow(charStyleCheckBox, FALSE);

} else {

SendMessage(charStyleCheckBox, BM\_SETCHECK, BST\_UNCHECKED, 0);

SendMessage(wordStyleCheckBox, BM\_SETCHECK, BST\_CHECKED, 0);

EnableWindow(charStyleCheckBox, TRUE);

}

break;

}

case VISUALIZE:

{

if (!data.empty()) {

isPaint = true;

InvalidateRect(hWnd, NULL, TRUE);

}

break;

}

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

if (isPaint) {

Types::VisualizationType type = IsDlgButtonChecked(hWnd, CHAR\_TYPE) ?

Types::VisualizationType::CHAR : Types::VisualizationType::WORD;

WinApi::getInstance().DrawVisualization(data, hdc, hWnd, type, inputType);

isPaint = false;

}

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

void GenerateData(HWND hWnd) {

int trackPos = SendMessage(amountTrackBar, TBM\_GETPOS, 0, 0);

Types::Type type = IsDlgButtonChecked(hWnd, INT\_TYPE) ? Types::Type::INT : Types::Type::STRING;

std::wstring fileName = WinApi::getInstance().OpenFileForm(hWnd);

if (!fileName.empty()) {

if (isBounds) {

DataGenerator\* generator = new DataGenerator(trackPos, type, bounds.first, bounds.second);

generator->generateAndWriteToFile(fileName.c\_str());

delete generator;

}

else {

DataGenerator\* generator = new DataGenerator(trackPos, type);

generator->generateAndWriteToFile(fileName.c\_str());

delete generator;

}

MessageBox(NULL, L"Data successfully writed in file", L"Success", MB\_OK);

}

}