Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Пояснительная записка

к курсовой работе по курсу
«Программирование»
на тему «Разработка программы сложной структуры методом нисходящего программирования».

Выполнил:
Студент группы 21ВВ1
Кулаков Дмитрий Алексеевич

Принял:
Токарев А.Н.

ОТТ
18.06.22

Пенза 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

_____ М.А. Митрохин

«____»_____20__г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Куликову Дмитрию Алексеевичу Группа 21ВВ1

Тема проекта: ***Разработка программы сложной структуры методом нисходящего программирования.***

Исходные данные (технические требования) на проектирование
Консольное приложение для обмена сообщениями между пользователями. Отправка и приём сообщений. Введение истории, сохранение истории на диск. Вставка на языке ассемблера.

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. **Расчетная часть**

Разработка программы.

2. **Графическая часть**

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. **Экспериментальная часть**

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "22" февраля

Дата защиты проекта "__" _____

Руководитель Токарев

Задание получил "22" февраля 20 22 г.

Студент Кулаков (Кулаков Д. А)

Оглавление

Введение 5
1 Постановка задачи 7
2 Выбор решения 8
3 Описание структуры программы 9
4 Описание функционала программы. 12
5 Руководство пользователя. 13
6 Разработка функции на языке Assembler. 17
7 Тестирование. 18
Заключение. 24
Приложение «Листинг программы» 25

Введение

Обмен письмами и телеграммами ушли далеко в прошлое. Сейчас уже практически невозможно представить свою жизнь без смартфонов и прочих гаджетов. Следовательно сообщения и переписки происходят электронным образом, с помощью таких технологий как мессенджеры, об одном из которых и есть эта работа.

Мессенджер – программа для мгновенной отправки сообщений. Предназначен для общения людей. Также можно общаться через соцсети, которые обладают более широким функционалом, но абсолютно не подходят для работы и задач, требующих повышенное внимание. Из-за большого потока контента в современных соцсетях человек не может сосредоточиться на своей задаче. Именно поэтому актуальны мессенджеры.

В период пандемии такие технологии показали свою важность и роль в жизни мира. Поэтому я считаю, что эта отрасль является перспективной для развития уже сейчас и будет многие последующие годы.

Сколько времени вы каждый день проводите в мессенджере? Час, два или даже больше? По данным исследования We Are Social в 2021 году, в среднем россияне проводят в интернете 7 часов 52 минуты в день. Большой процент этого времени приходится на мессенджеры, и столь популярные приложения для общения не могут не влиять на наш образ жизни, коммуникации и в целом на общество и культуру.

Пандемия лишь ускорила эти изменения: сегодня люди вынуждены соблюдать меры осторожности, социальную дистанцию и реже обращаться в живую. Чтобы оставаться на связи, нам на помощь приходят технологии.

Нельзя оставить без внимания и последние события в мире. Сейчас в нашей стране проводится политика импортозамещения для укрепления суверенитета. Проанализировав статистику отечественных программ и информационных систем, можно сделать вывод, что сейчас в России есть только 2 мессенджера Telegram и VK, которые особо популярны и по факту являются монополистами в этой сфере технологий, что делает неконкурентоспособными более новые разработки. Среди отечественнных разработок я также изучил:

Куку-AU – мессенджер с забавным названием, тем не менее, предназначенный для серьезных целей. Авторами позиционируется как приложение для общения между российскими бизнесменами и соответствующими службами;

Rugram – мессенджер с патриотическим «привкусом» от кремлевского молодежного движения. Код основан на открытой части программы Telegram, имеются специфические стикеры патриотической направленности;

Avirton – мессенджер, специализирующийся исключительно на управлениях финансами. Например, переводы денег, покупка-продажа акций. Разработчик Дмитрий Караев;

FireChat – любопытная разработка от выпускника МГУ Станислава Шалунова. Данный мессенджер позволяет осуществлять общение даже при полностью отсутствующей сотовой связи и Интернете. Прием и передача сообщений производятся в радиусе до 70 метров напрямую между устройствами, на которых установлено приложение.

Исходя из вышесказанного, я решил проверить свои возможности в реализации похожего программного продукта. Мой результат работы представляет собой консольное приложение, предназначенное для обмена и передачи сообщений в текстовом формате. В дальнейшем эти сообщения сохраняются в файл на неограниченный срок. Так и получается база данных из файлов «чатов», что позволяет просматривать историю сообщений.

1 Постановка задачи

Необходимо разработать консольное приложение для обмена сообщениями между пользователями. Программа должна отправлять и принимать сообщения, вести историю сообщений, сохраняя их на диск.

Нужно разработать интуитивно понятный интерфейс пользователя, определить пункты названия меню. Изучить работу с файлами и с сетевыми интерфейсами. Это необходимо для чёткого понимания функциональности программы.

Приложение должно отвечать следующим требованиям:

1. Многомодульность. Программа должна быть поделена на логические модули. Это упростит дальнейшее совершенствование программы, а также позволит локализовать ошибки.
2. Использование сложных типов данных: массивов, структур, объектов и т.д. Для разработки консольного мессенджера необходимо подробно изучить принципы работы с массивами и объектами для реализации алгоритмов хранения сообщений и взаимодействия с сервером.
3. Режим работы видеосистемы: текстовый/графический. Необходимо определиться с типом управления, видом интерфейса и изучить приёмы реализации графики.
4. Устройство ввода информации: клавиатура/мышь.
5. Пользовательский интерфейс должен быть построен на основе меню или панели инструментов. Необходимо реализовать пользовательское меню команд, с заставкой.
6. Операционная система: MS Windows. При разработке использовать среду MS Windows с системой программирования MS Visual Studio.
7. Язык программирования: Си и Ассемблер.

2 Выбор решения

2.1 Определение необходимых модулей программ
Разработанная программа состоит из двух модулей:

1. Библиотека libcurl-d.dll
2. ibook.exe

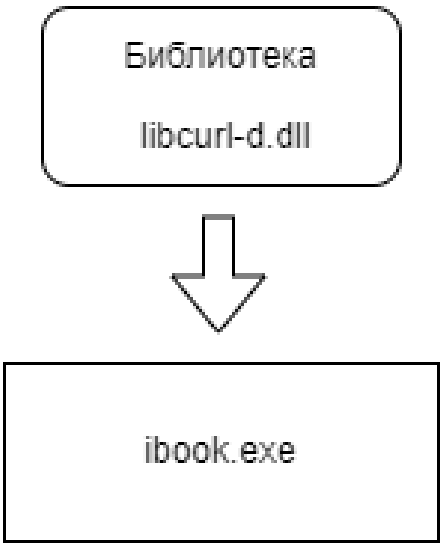


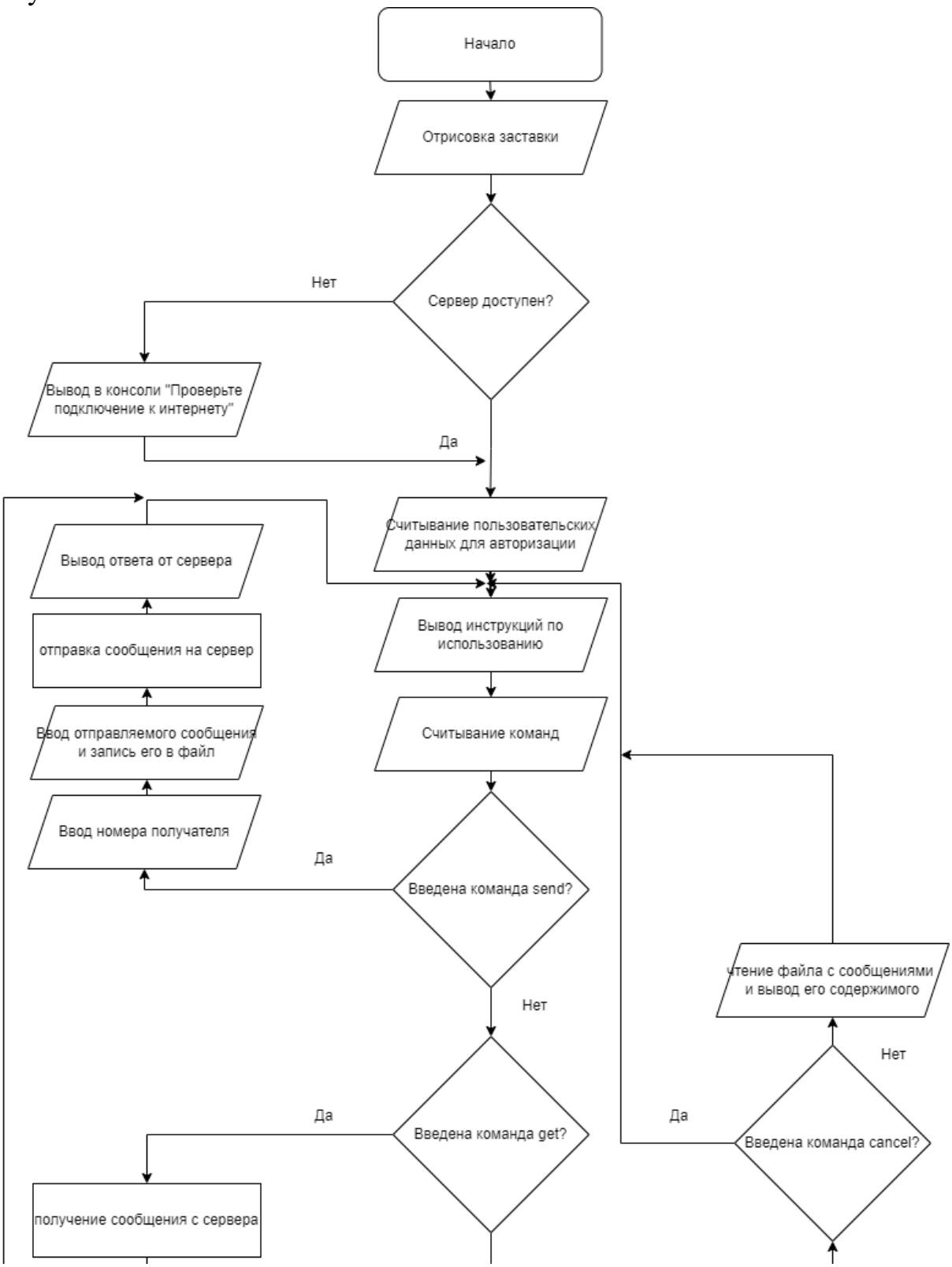
Рисунок 1 – структура модулей программы.

Библиотека libcurl-d.dll необходима для реализации взаимодействия программы с сетевыми интерфейсами, была использована для выполнения данной курсовой работы и не модифицировалась. Библиотека предоставляет богатый набор функций и позволяет: совершать get и post запросы на сетевые ресурсы, получать коды ответа от сервера, ответы в формате JSON и страницы html.

Приложение ibook.exe является основным модулем данной курсовой работы и использует функции из библиотеки libcurl-d.dll, удовлетворяя требование модульности курсовой работы. Управление приложением осуществляется путём ввода команд с клавиатуры. При запуске приложение попросит вас ввести номер телефона для авторизации и выведет инструкцию по использованию, после чего мессенджер готов к работе. Программа предназначена для отправки и принятия сообщений между устройствами под управлением ОС Windows. Сообщения предаются на сервер в виде JSON строк POST запросами. Приходят сообщения в качестве ответа от сервера в JSON формате. Приложение даёт возможность просматривать историю сообщений. Вся переписка записывается в файл с именем получателя.

3 Описание структуры программы

Файл «ibook.cpp» - главный файл. Функция «main» – главная функция, в которой вызываются функции, определяющие проверку сервера и дальнейшую работу с сообщениями. Структура программы показана на рисунке 2.



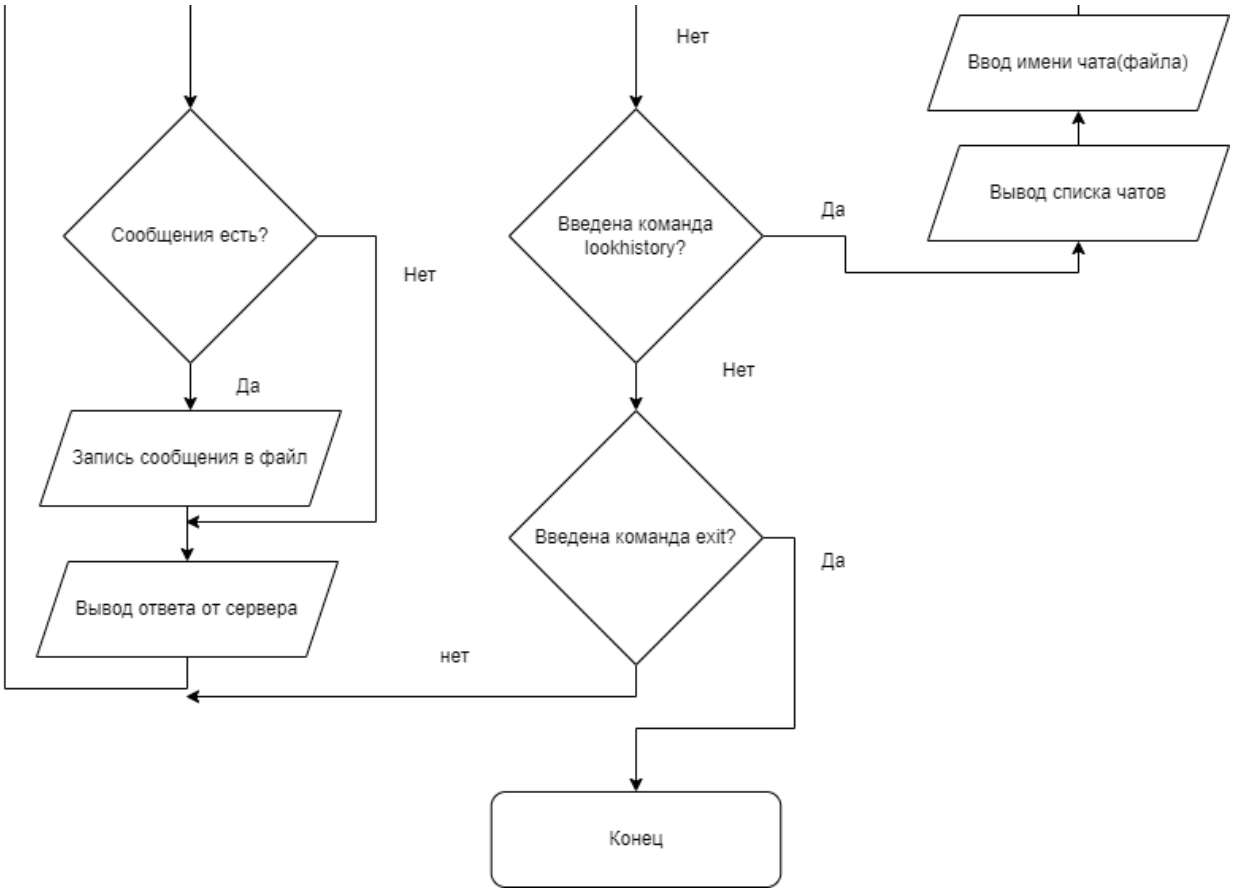


Рисунок 2 – структура программы.

Заголовочный файл «Configuration.h» - файл быстрой настройки, в можно поменять адрес сервера, названия пунктов меню и команд. «ibook.cpp» - главный файл программы, содержит функции: screensaver, expectationCommand и main.

main – содержит вызовы основных функций.

screensaver - рисует заставку посредством вывода строк через printf().

expectationCommand – ожидает true от функций ввода команд и запускает отправку, получение и просмотр сообщений. Также содержит код для вывода инструкции по использованию.

«CheckServer.h» - содержит объявление функций из CheckServer.cpp. CheckServer.cpp – файл в котором реализованы основные функции по работе с сервером: checkServer, getResponseData.

checkServer – функция в которой проводится инициализация объекта библиотеки curl и совершение get запроса на сервер для проверки доступности.

getResponseData – функция записи ответа сервера.

«Messanger.h» - содержит объявление функций из «Messanger.cpp». «Messanger.cpp» - файл проверки вводимых команд в консоли и отправки POST запросов на сервер. Содержит функции: get_nickname,

send_message_comand, get_message_comand, get_history, sendMessage, getMessage.

Функций ввода команд:

get_nickname – реализация ввода телефона пользователя.

send_message_comand – сравнивает образец команды из Configuration.h с введённой строкой в консоли. Возвращает true в случае, если ввели команду отправки сообщений.

get_message_comand – сравнивает образец команды из Configuration.h с введённой строкой в консоли. Возвращает true в случае, если ввели команду проверки новых сообщений.

get_history – сравнивает образец команды из Configuration.h с введённой строкой в консоли. Возвращает true в случае, если ввели команду просмотра истории сообщений.

Функции отправки POST запросов:

sendMessage – отправляет запрос на сервер в теле которого JSON строка (возвращает функция send_message_JSON). Ожидаемым результатом выполнения функции является отправка сообщения на сервер.

getMessage – отправляет запрос на сервер в теле которого JSON строка (возвращает функция get_message_JSON). Ожидаемым результатом выполнения функции является ответ от сервера, в теле которого будет сообщение отправителя или информация об отсутствии сообщений.

«FormatJSON_String.h» - содержит объявления функций из «FormatJSON_String.cpp». «FormatJSON_String.cpp» - файл реализации создания JSON строк. Содержит send_message_JSON, get_message_JSON функции и объявление addStr.

send_message_JSON – функция формирования JSON строки для функции отправки сообщений (sendMessage).

get_message_JSON – функция формирования JSON строки для функции проверки новых сообщений (getMessage).

Файл «addStr.asm» - реализованная в ассемблере функция addStr, выполняет сложение 3-х строк.

«File_manager.h» - содержит объявление функций из «File_manager.cpp». «File_manager.cpp» – файл работы с файлами, содержит функции save_message, readMessages, get_list_file.

save_message – функция сохранения сообщения в файл.

readMessages – функция чтения сообщений из файла.

get_list_file – функция получения списка чатов (сообщений).

4 Описание функционала программы.

Функция «main» – главная функция. В ней происходит запуск всех важных функций. Сначала вызывается функция screensaver() - для отрисовки заставки программы. Затем вызываются функции checkServer() - для проверки подключения к интернету и доступности сервера, get_nickname – для получения номера пользователя и его авторизации.

Далее в функции expectationCommand() выводится инструкция по использованию программы, в которой указаны все команды. В ней реализованы несколько условий для проверки функций команд (send_message_command(), get_message_command(), get_history()) при выполнении которых, вызываются sendMessage(), getMessage(), get_list_file() соответственно.

Доступные команды:

send – после ввода этой команды, от пользователя потребуется ввести номер получателя и сообщение, заменив пробелы на нижнее подчёркивание, после чего сообщение отправится на сервер.

get – проверяет наличие новых сообщений.

lookhistory – показывает чаты со старыми сообщениями, после ввода этой команды необходимо ввести имя интересуемого чата для чтения сообщений или ввести команду cancel.

exit – завершение работы программы.

Если пользователь набрал команду send, то вызывается функция sendMessage(), которая отправляет строку из send_message_JSON() на сервер.

Пример JSON строки:

```
 {
 "request": {
        "command": " +79270969849",
        "phone": "+79683949871"
        },
 "version": "message_from_+79270969849"
 }
```

Ответ от сервера придёт в виде обычной строки, которая сразу же будет выведена в консоль.

Также после ввода сообщения будет вызвана функция save_message(), в которой реализовано сохранение сообщения в файл. Если пользователь набрал команду get, то вызывается функция getMessage(), которая отправляет строку из get_message_JSON() на сервер.

Пример JSON строки:

```
{
"request": {
        "command":"getMessage",
        "phone": "+79270969849”
        },
"version": ""
}
```

Ответ от сервера придёт в виде обычной строки, которая сразу же будет выведена в консоль.

Если же ответ будет начинаться с номера телефона, то будет вызвана функция save_message(), в которой реализовано сохранение сообщения в файл.

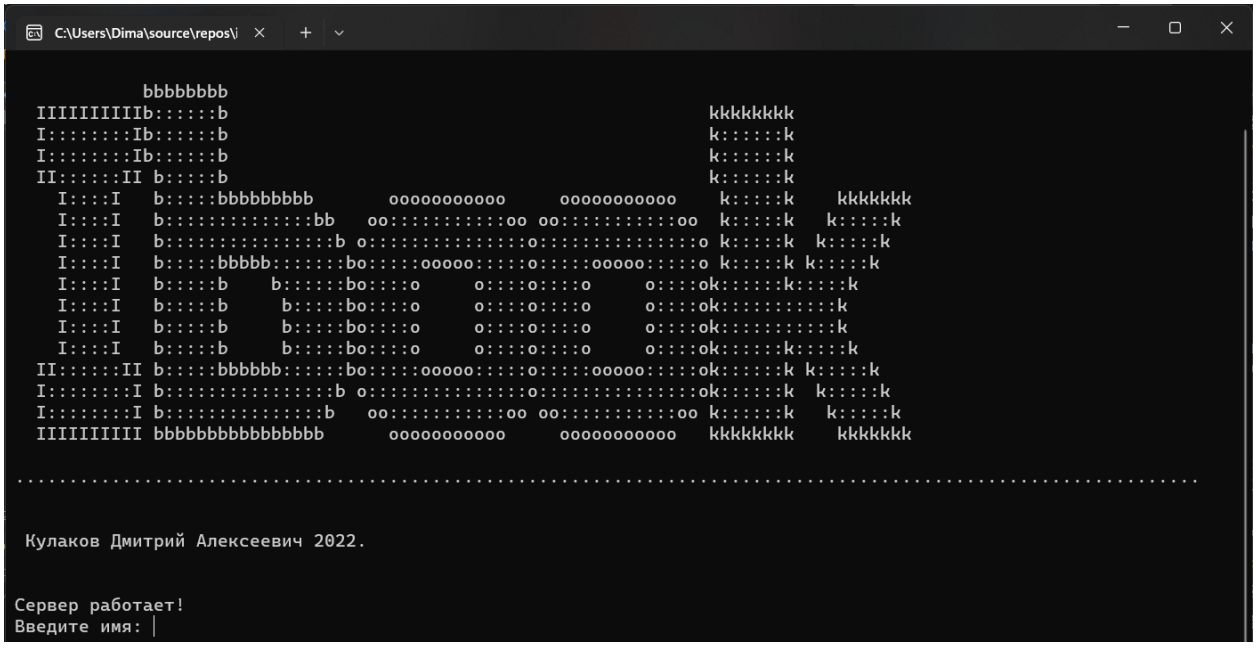
Чтение файлов будет производится в функции readMessages().

После каждого выполнения команд вызывается функция expectationCommand(), вследствие чего мессенджер запускается заново до тех пор, пока не будет введена команда exit.

5 Руководство пользователя.

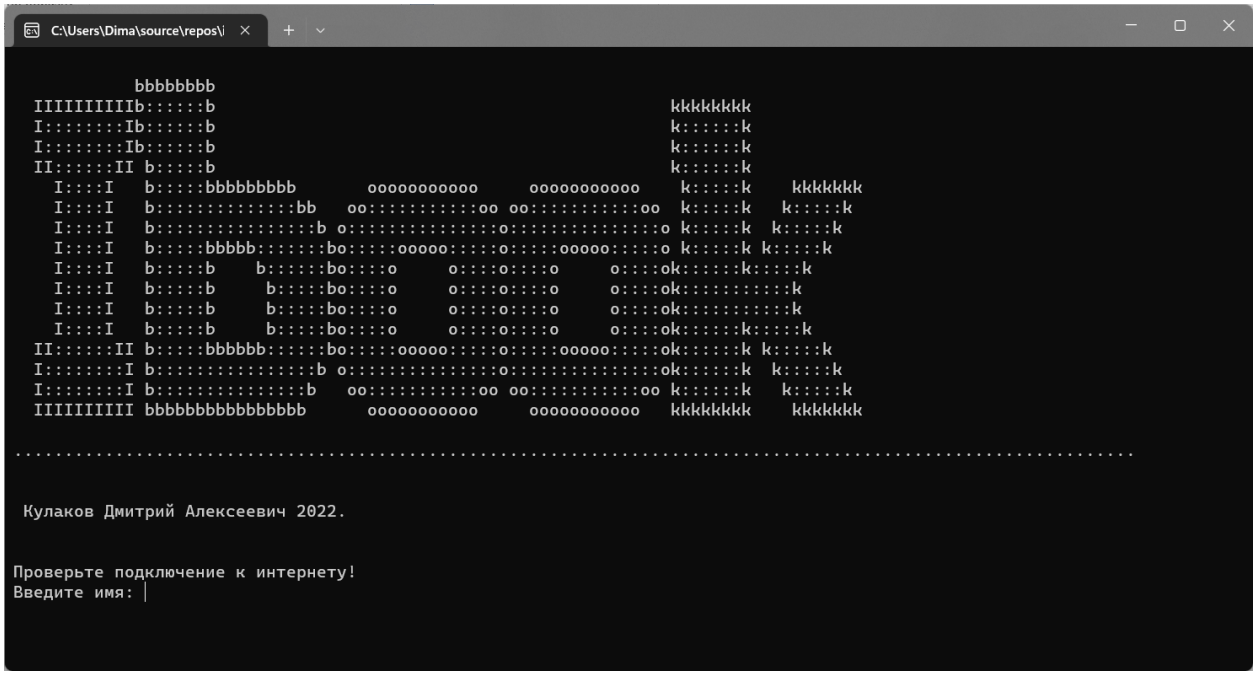
«Консольный мессенджер ibook» - консольный мессенджер с сохранением сообщений в файл. При загрузке программы отобразится заставка, с информацией о доступности сервера и с просьбой о вводе вашего номера телефона для авторизации.

В случае успешной проверки сети:

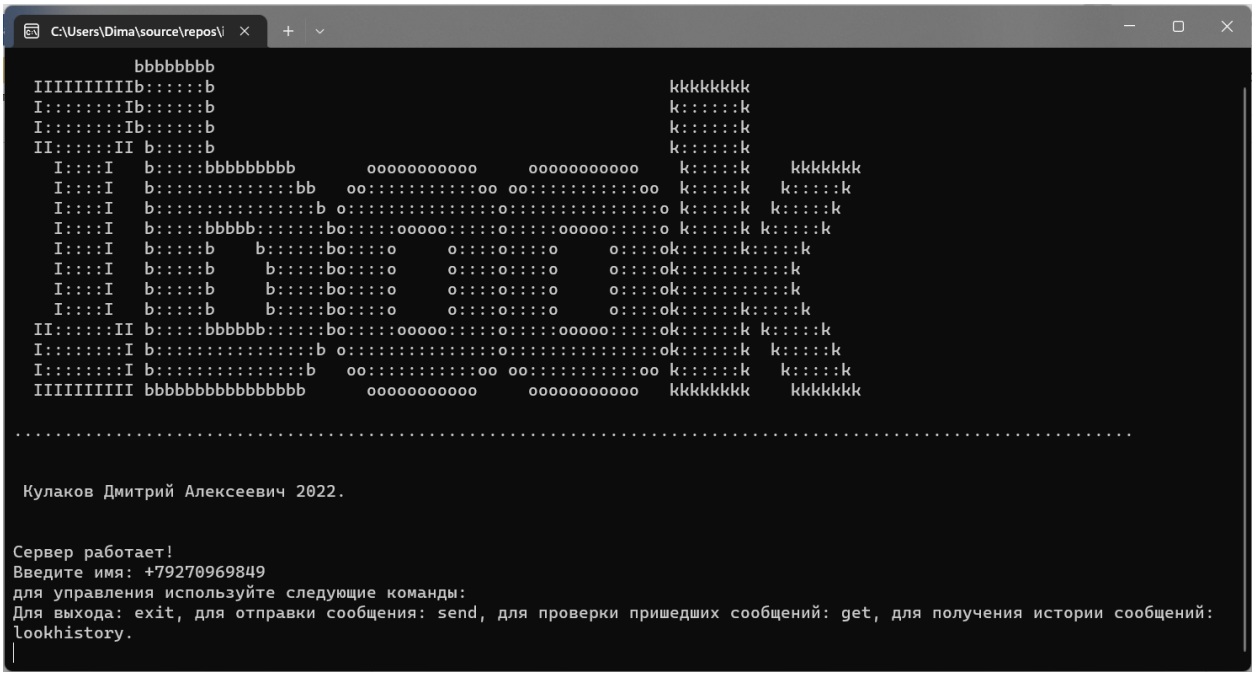


В случае если сервер выключен или его url в Configuration.h не действителен:

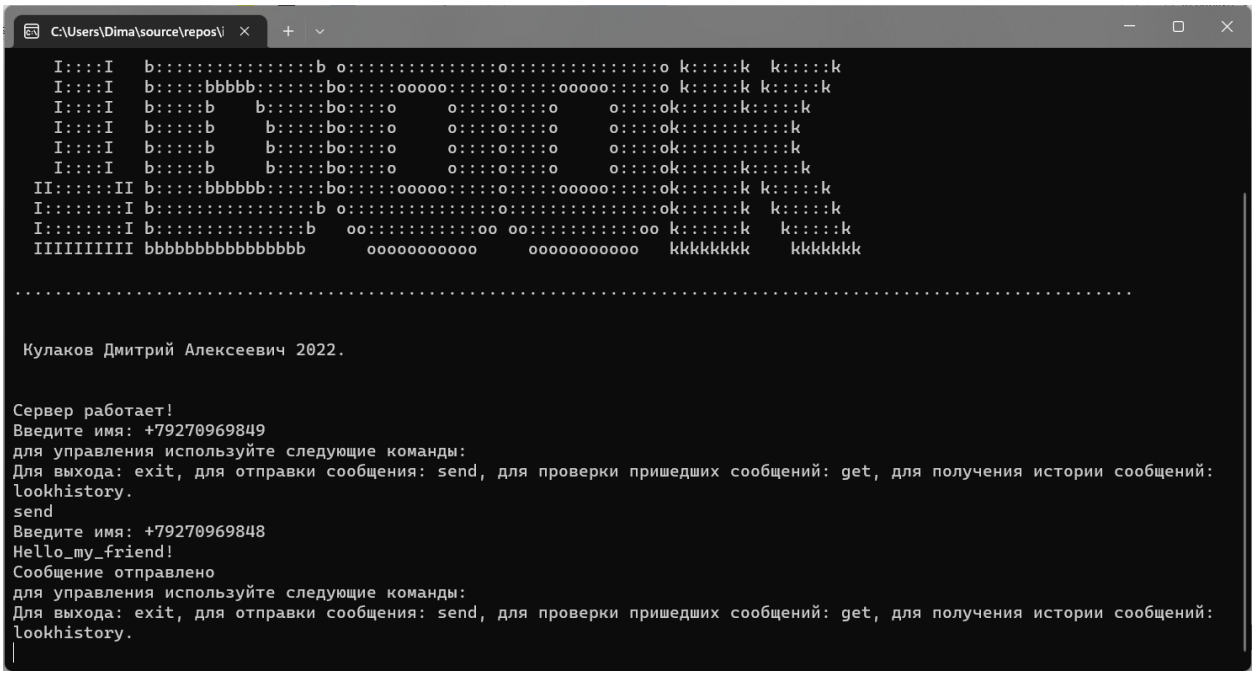
В случае если компьютер не подключен к интернету:



После ввода вашего номера, будут выведены доступные команды.

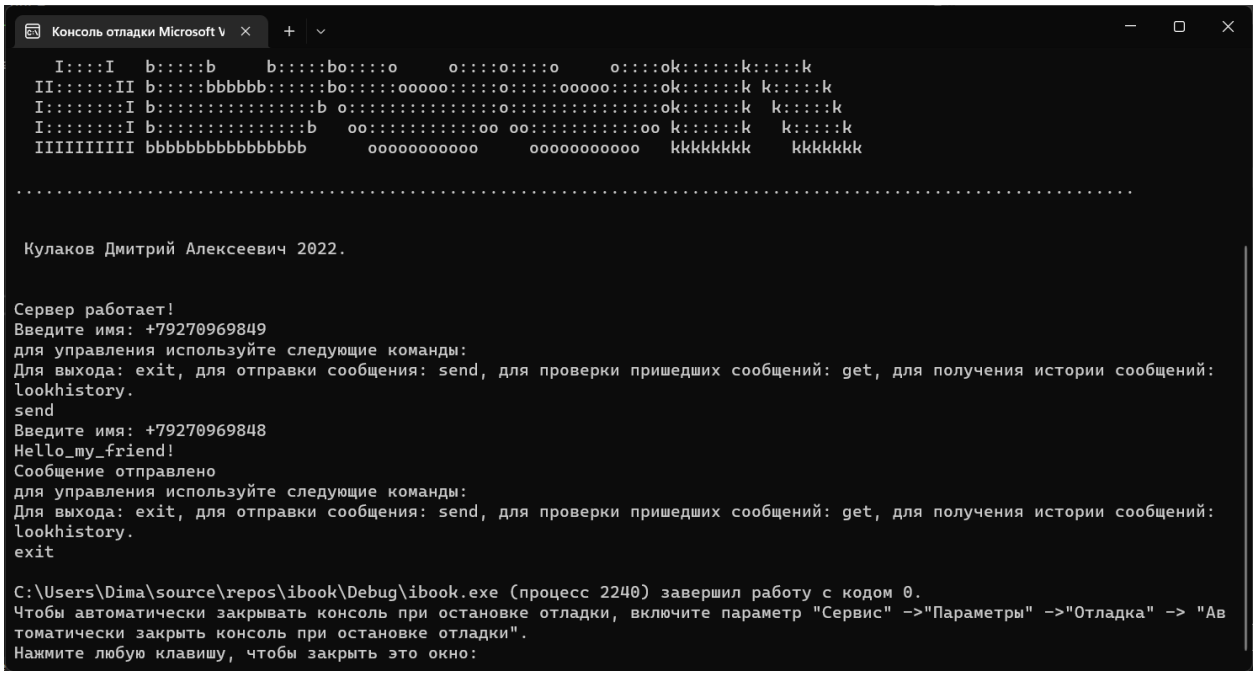


После ввода команды send введите номер получателя и сообщение которые вы хотите ему отослать (все пробелы замените на нижние подчёркивания).



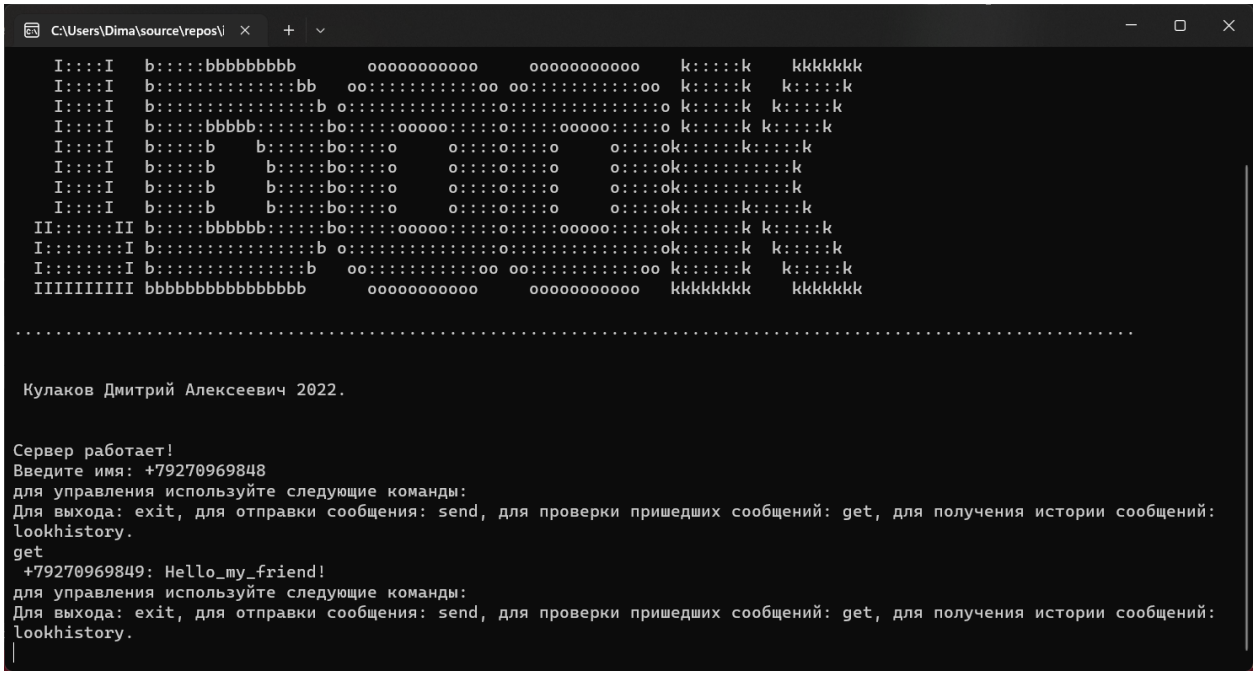
После отправки выведется ответ от сервера: «Сообщение отправлено» и инструкция.

Выход:

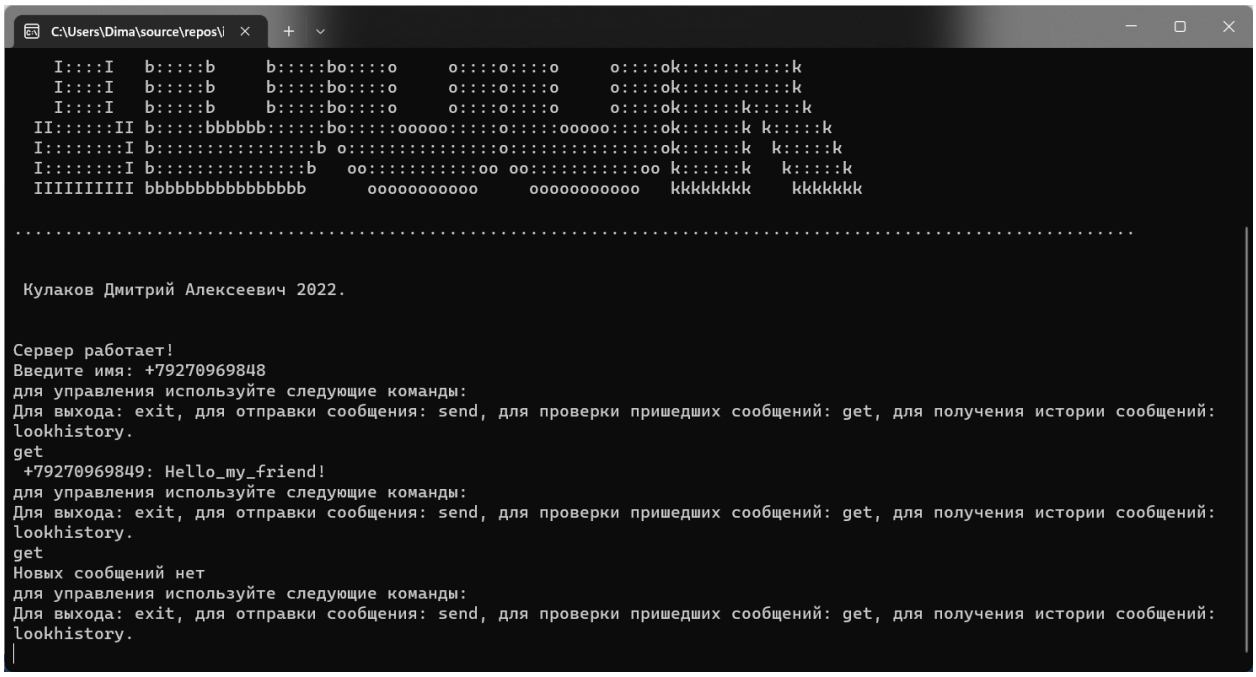


Для получения и проверки новых сообщений введите get:

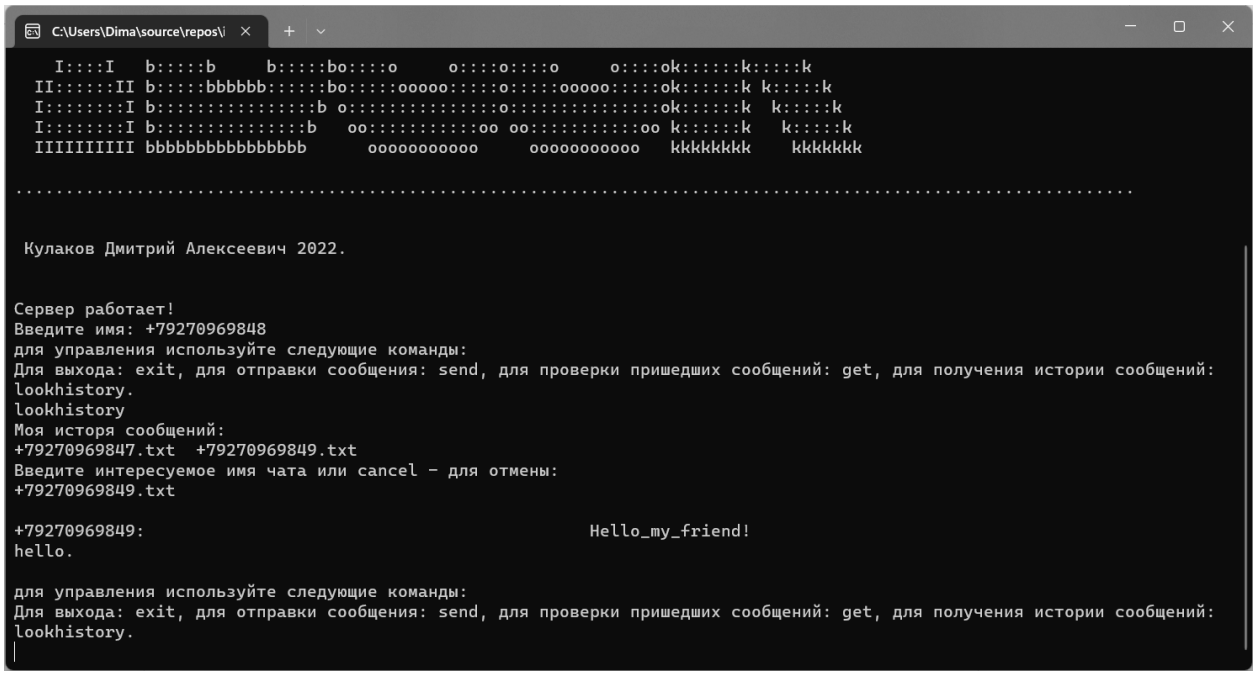
Ответ сервера в случае, если новые сообщения есть:



В случае, если новых сообщений нет:



Просмотр истории сообщений произведите с помощью команды lookhistory. Для просмотра нужно ввести имя файла:



Ваши сообщения будут выводится слева без номера.

6 Разработка функции на языке Assembler.

В своей работе я использовал ассемблер для перезаписи и сложения строк.

Например:

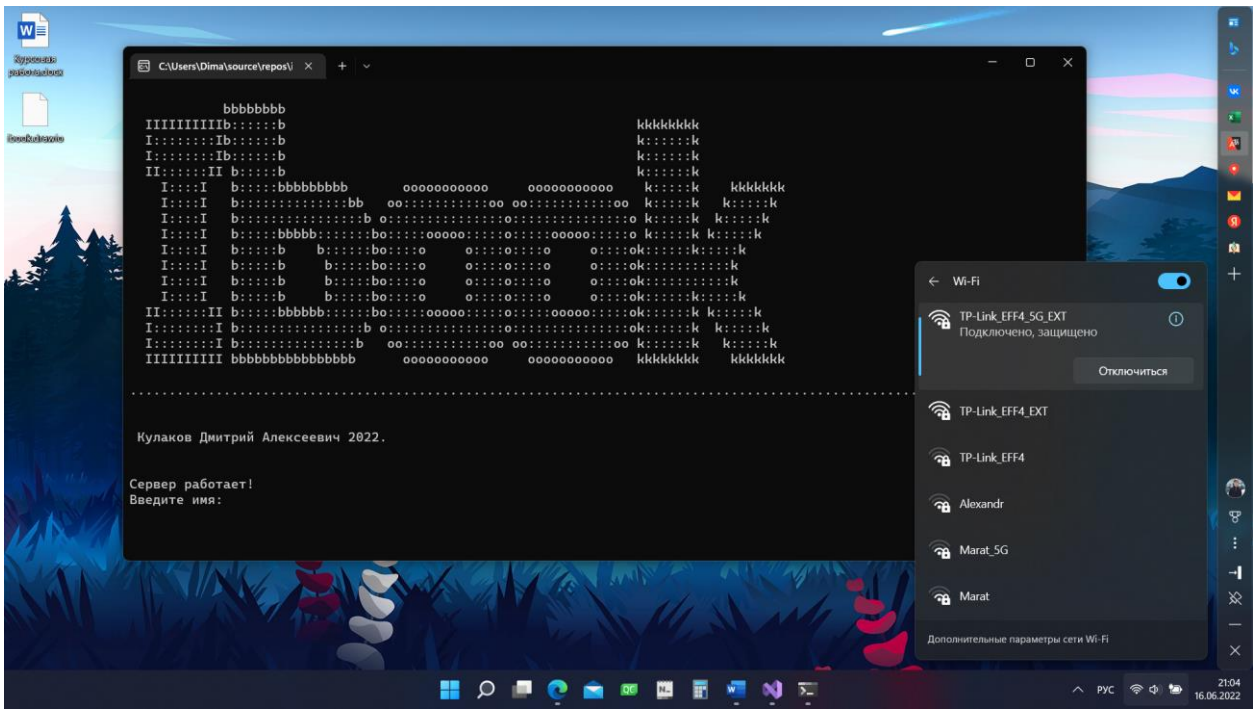
```
mov edi,stroka1                                    ; stroka1 – адрес прибавляемой
строки
lea esi, result

addstr1:                                           ;запись данных из места в
оперативной памяти по адресу edi в esi
    mov al, byte ptr[edi]
    mov [esi],al
    inc edi
    inc esi                                        ;инкриминтирование для
следующего символа
loop addstr1
```

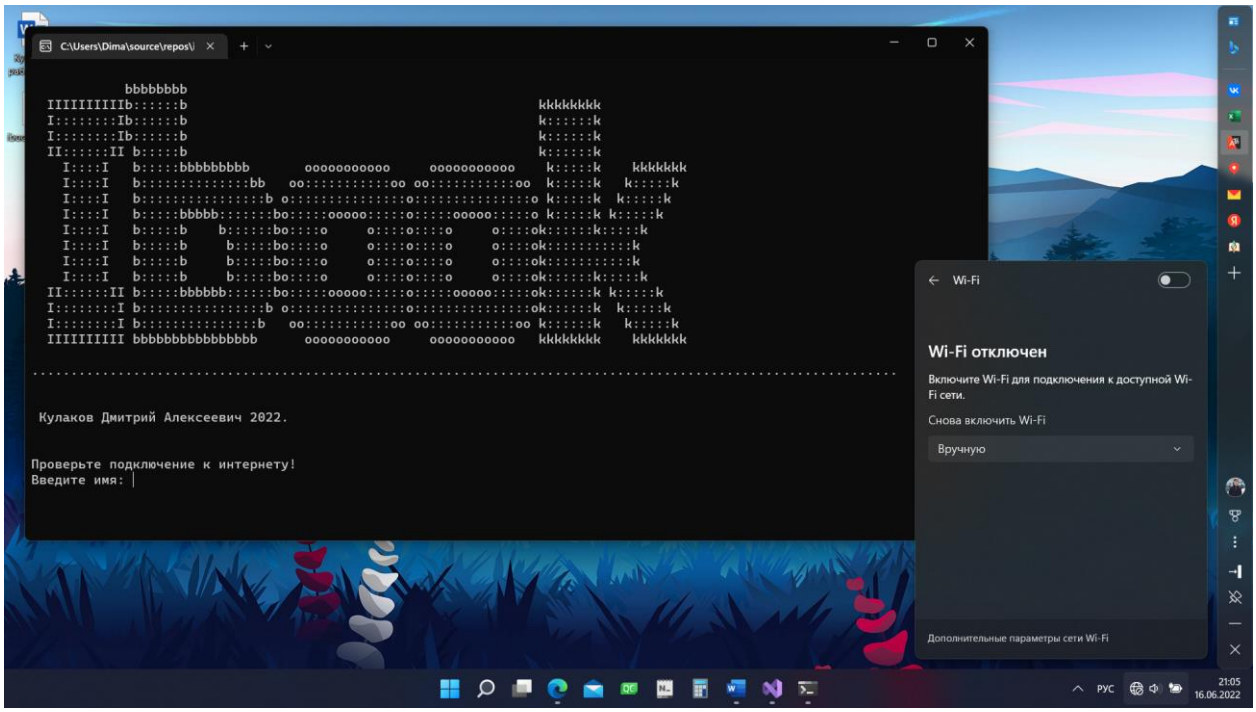
Мной были использованы следующие ассемблерные инструкции: инструкция mov выполняет копирование содержимого источника в приемник; инструкция lea выполняет запись адреса источника в оперативной памяти в регистр; инструкция inc выполняет сложение содержимого приемника с 1; инструкция loop передаёт управление к указанной метке, пока в регистре ecx не будет 0; macro – задаёт макрос, содержимое которого будет подставлено в код программы.

7 Тестирование.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.

1. Тестирование на автоматическое определение состояния сети.

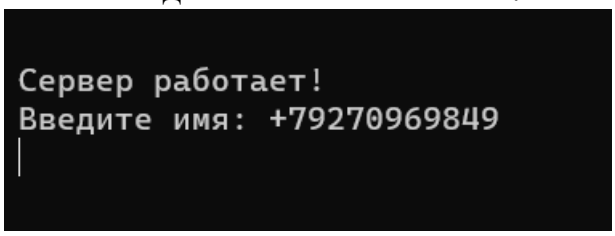


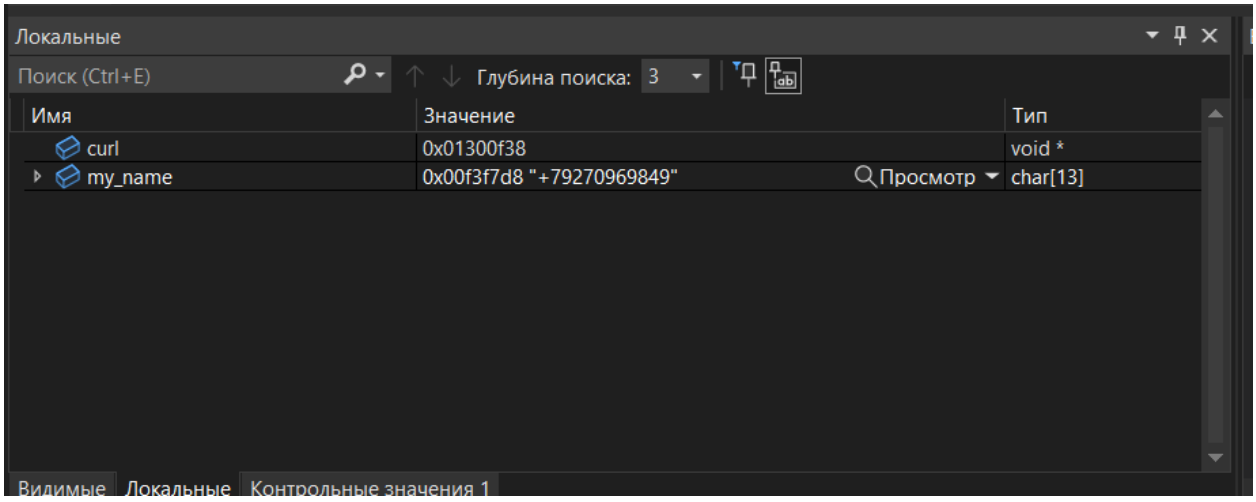


При запуске программы в консоли должно быть выведено о состоянии сети.

Определение проходит корректно.

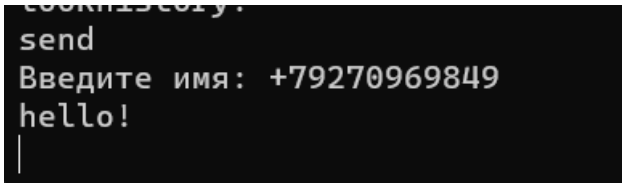
1. Тестирование на корректность приёма строк и команд в консоли
   Ввод имени пользователя:

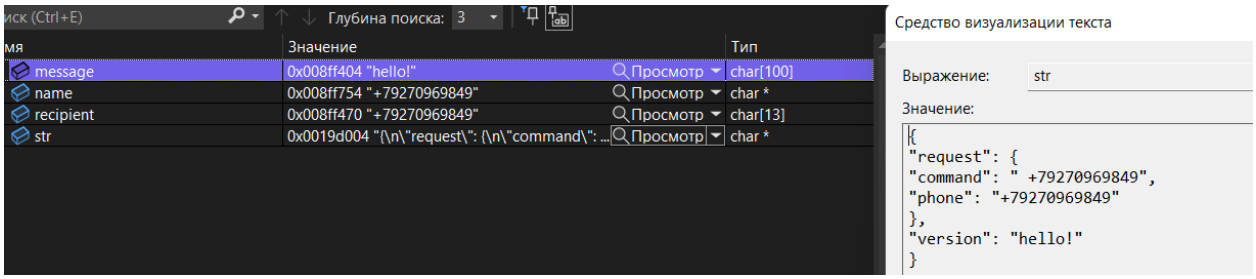




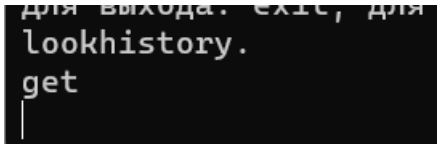
Ввод прошёл корректно

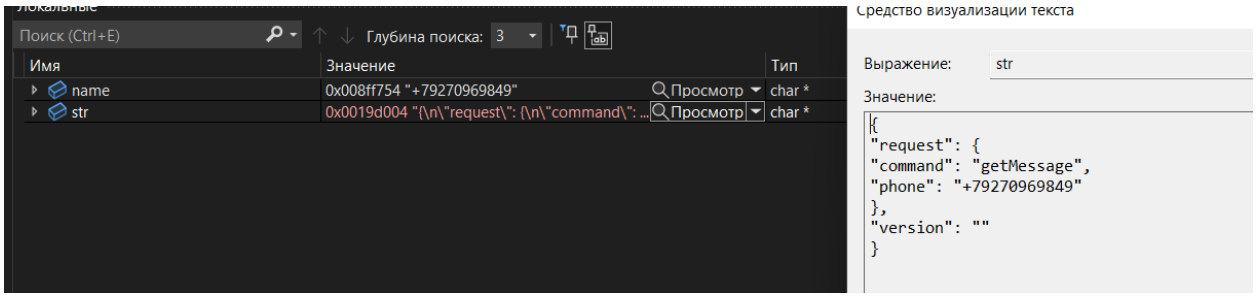
2. Проверка корректности формирования JSON строк для дальнейшей их отправки на сервер:
   Тест команды send:





Тест команды get:

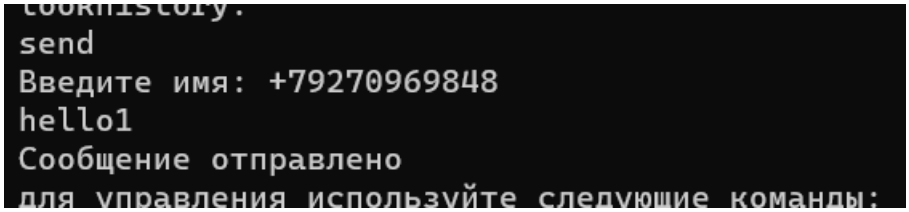




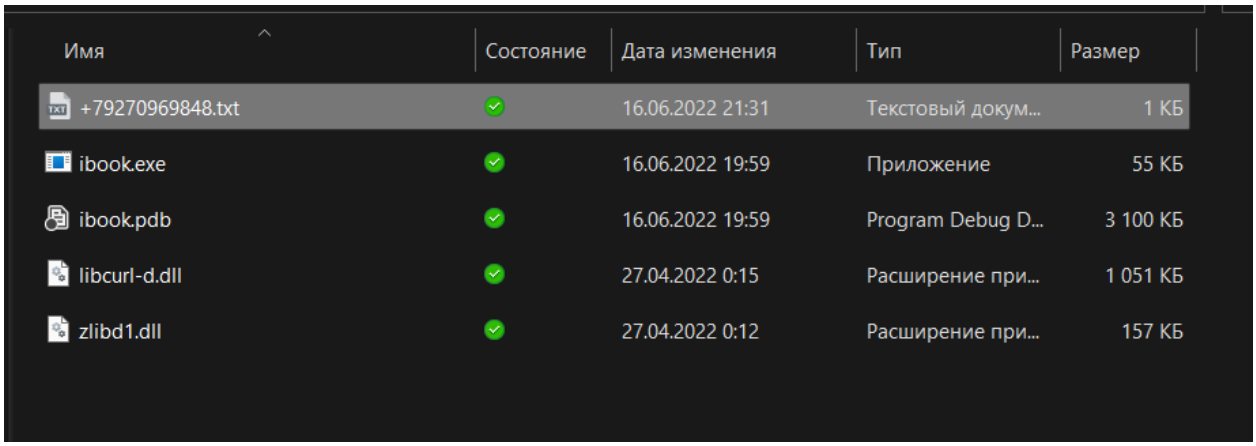
Команды работают верно, JSON строки в теле запросов, производимых программой, верны.

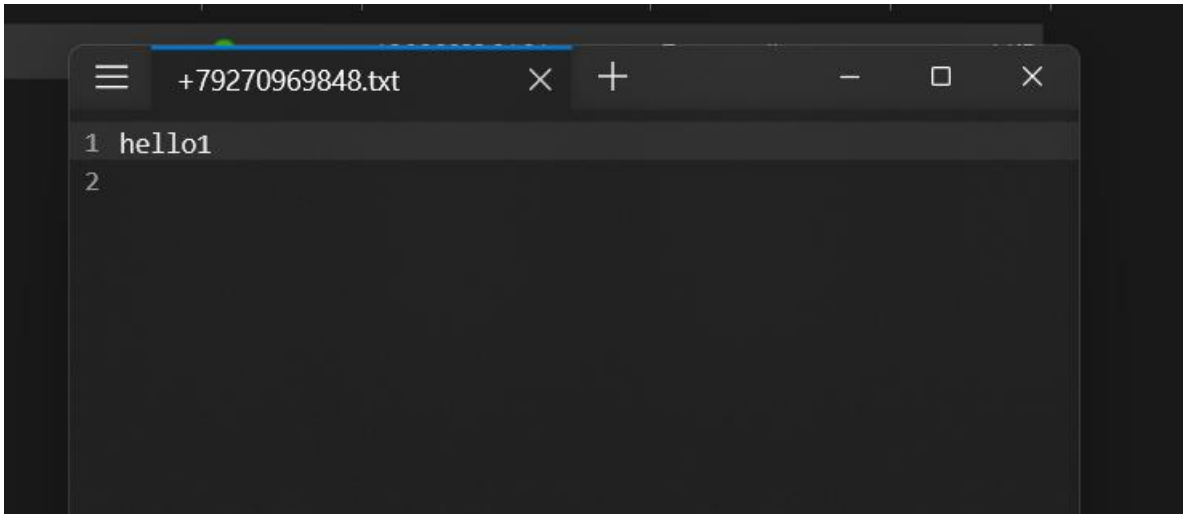
3. Проверка корректности записи файлов сообщений.

   Команда send:



Результат:

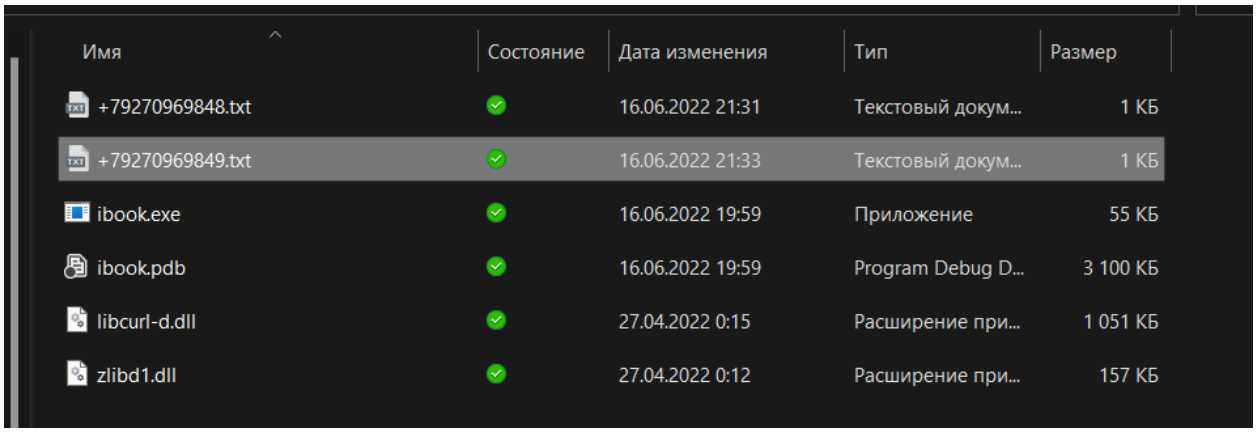


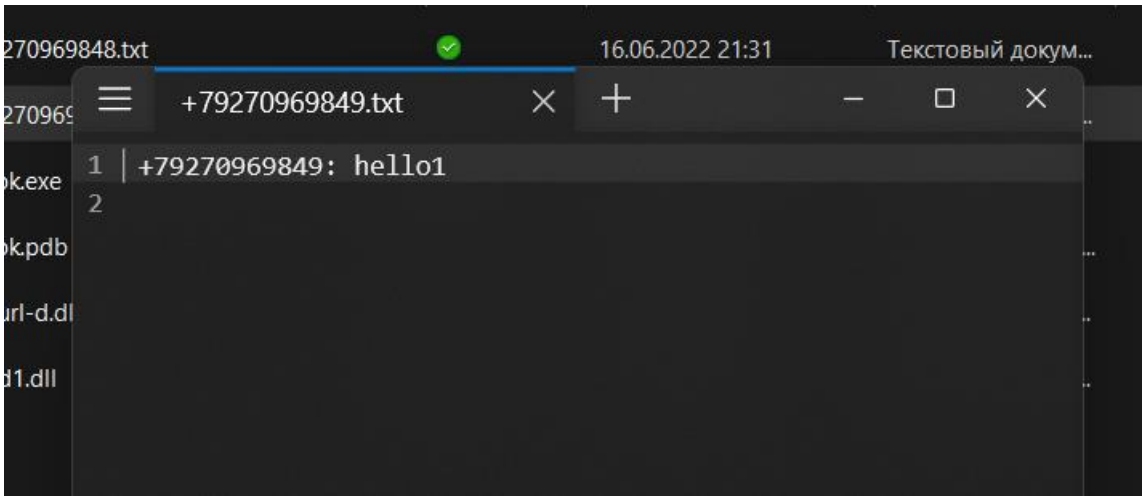


Команда get:

```
для выхода: exit, для отправки сообщения: send, для
lookhistory.
get
 +79270969849: hello1
для управления используйте следующие команды:
Для выхода: exit, для отправки сообщения: send, для
lookhistory.
```

Результат:

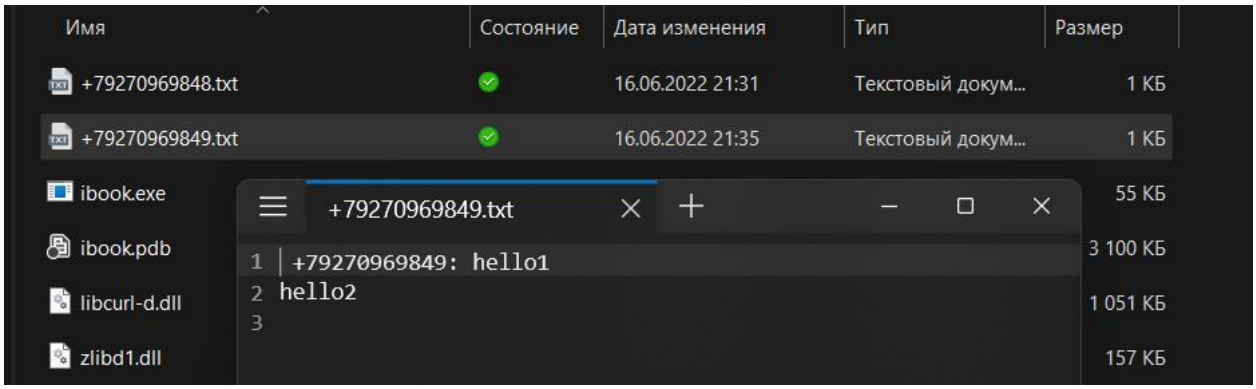




Отправка повторного сообщения:

```
Введите имя: +79270969848
для управления используйте следующие ко
Для выхода: exit, для отправки сообщени
lookhistory.
get
 +79270969849: hello1
для управления используйте следующие ко
Для выхода: exit, для отправки сообщени
lookhistory.
send
Введите имя: +79270969849
hello2
Сообщение отправлено
```

Результат:



Запись сообщений в файл работает корректно.

В таблице 1 представлена сводная информация по результатам тестирования.

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Автоматическое определение состояния сети.	Верно
Ввод команд с клавиатуры	Программа корректно принимает команды	Верно
Проверка asm функции сложения строк	Сложение строк производится верно	Верно
Проверка отправки сообщений	Сообщения сохраняются на сервере в файл	Верно
Проверка принятия сообщений	Сообщения доходят до адресата	Верно
Проверка сохранения истории сообщений	Сообщения сохраняются в файл	Верно

В результате тестирования программы было выявлено, что программа функционирует успешно.

Заключение.

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Были освоены принципы работы с сервером при помощи библиотеки «curl». Изучены способы создания и обработки POST и GET запросов. Получены знания и навыки по работе с файлами. Получены базовые навыки программирования на языке C и Ассемблера, а также некоторых функций из C++. Изучены способы создания и интеграции с C кодом asm файлов. Получены навыки отладки и тестирования программ.

В рамках данной курсовой работы был написан консольный мессенджер. Данная программа не требует большого количества системных ресурсов и имеет систему команд для удобства и простоты использования.

В дальнейшем, есть идеи по усовершенствованию программы: сделать поддержку сообщений на русском, убрать команду get и проверять сообщения автоматически во втором потоке, прейти на графический интерфейс, изучить и использовать web сокеты для повышения производительности, создать мобильную версию для телефонов и планшетов на языке java, переписать сервер добавив возможность отправки сразу нескольких сообщений (сейчас сообщения записываются на сервере в файл с именем адресата и в случае новых перезаписывают файл).

Приложение «Листинг программы»

Файл «ibook.cpp»

```
/*Консольный мессенджер ibook 2022
*/
#define _CRT_SECURE_NO_WARNINGS
#include "CheckServer.h"
#include "Configuration.h"
#include "Messanger.h"
#include "File_manager.h"

void screensaver() {
	printf("\n            bbbbbbbb\n");
	printf("  IIIIIIIIIb::::::b
kkkkkkkk\n");
	printf("  I::::::::Ib::::::b
k::::::k\n");
	printf("  I::::::::Ib::::::b
k::::::k\n");
	printf("  II::::::II b:::::b
k::::::k\n");
	printf("    I::::I   b:::::bbbbbbbbb       oooooooooooo     oooooooooooo
k:::::k    kkkkkkk\n");
	printf("    I::::I   b::::::::::::::bb   oo::::::::::::oo oo::::::::::::oo
k:::::k   k:::::k\n");
	printf("    I::::I   b::::::::::::::::b o::::::::::::::o::::::::::::::o
k:::::k  k:::::k\n");
	printf("    I::::I   b:::::bbbbb:::::::bo:::::ooooo:::::o:::::ooooo:::::o
k:::::k k:::::k\n");
	printf("    I::::I   b:::::b    b::::::bo::::o     o::::o::::o
o::::ok::::::k:::::k\n");
	printf("    I::::I   b:::::b     b:::::bo::::o     o::::o::::o
o::::ok::::::::::k\n");
	printf("    I::::I   b:::::b     b:::::bo::::o     o::::o::::o
o::::ok::::::::::k\n");
	printf("    I::::I   b:::::b     b:::::bo::::o     o::::o::::o
o::::ok::::::k:::::k\n");
	printf("  II::::::II
b:::::bbbbbb::::::bo:::::ooooo:::::o:::::ooooo:::::ok::::::k k:::::k\n");
	printf("  I::::::::I b::::::::::::::::b
o::::::::::::::o::::::::::::::ok::::::k  k:::::k\n");
	printf("  I::::::::I b::::::::::::::b   oo::::::::::::oo oo::::::::::::oo
k::::::k   k:::::k\n");
	printf("  IIIIIIIIII bbbbbbbbbbbbbbbb      oooooooooooo     oooooooooooo
kkkkkkkk    kkkkkkk\n\n");

	for (int i = 0; i <= 110; i++) {
		printf(".");
	};
	use_Unicode;
	printf("\n\n\n Кулаков Дмитрий Алексеевич 2022.\n\n\n");

}

void expectationCommand(CURL* curl, char* my_name) {
	char command[20];
	use_Unicode;
	printf(manual);
	printf(manualexCommand);
	printf(exitCommand);
	printf(manualMessangerCommand);
```

```
	printf(sendMessageCommand);
	printf(manualMessangerGetCommand);
	printf(getMessageCommand);
	printf(manualhistory);
	printf(history);
	printf(".\n");
	scanf("%s", command);

	if (!send_message_command(command)) {		// если введена команда sendMessageCommand (смотреть в Configuration.h) запустить функцию отправки сообщения
		sendMessage(curl, my_name);
	}
	if (!get_message_command(command)) {		// если введена команда getMessageCommand (смотреть в Configuration.h) запустить функцию проверки сообщений
		getMessage(curl, my_name);
	}

	if (!get_history(command)) {			// если введена команда history (смотреть в Configuration.h) запустить функцию проверки истории сообщений
		get_list_file();
		expectationCommand(curl, my_name);	// повторный запуск
	}
}

void main() {
	screensaver();
	char my_name[13];
	CURL* curl = checkServer();				// вызов функции проверки сервера
	use_Unicode;
	get_nickname(my_name);
	expectationCommand(curl, my_name);
	curl_easy_cleanup(curl);				// завершение работы curl
	}
```

Файл «ibook.h»

```
#include <curl/curl.h>
#pragma once
	void expectationCommand(CURL* curl, char* name);
```

Файл «CheckServer.cpp»

```
#include <curl/curl.h>
#include "Configuration.h"
#include "File_manager.h"

void getResponseData(char* data, size_t size, size_t n) {
	/*Функция записи данных при их приёме с помощью опции CURLOPT_WRITEFUNCTION, data – указатель на принимаемые данные, size – размер принимаемого блока данных, n – колличество блоков данных*/
	char* response;
	response = (char*)(data), (size * n);
	response[strlen(response) - 5] = 0;
		// удаление лишнего сивола приходящего в ответе сервера (для будущей корректной записи в файл)
	if (response[1] == '+') {
		save_message(response + 1, response);
	//сохранение полученных сообщений
	}
	use_UTF_8;
	printf("%s", response);
			// вывод ответа сервера
```

```
}

CURL* checkServer() {
	/*Эта функция инициализирует объект curl и проверяет доступность сервера*/
	use_Unicode;
	CURL* curl = curl_easy_init();
		// начинает easy(простую) curl-сессию, возвращает её дескриптор, в
случае неудачи NULL.
	if (curl) {
				// та самая проверка на NULL
		curl_easy_setopt(curl, CURLOPT_URL, url);
	// задаём url сервера с которым и будем работать
		curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, getResponseData); //
вызов функции getResponseData для записи полученного ответа от сервера
		if (curl_easy_perform(curl)) {
		// если curl_easy_perform(curl) возфращает 0, то соединение прошло
удачно
			printf("Проверьте подключение к интернету!\n");
		}
		// коды ошибок смотрел тут: https://curl.se/libcurl/c/libcurl-
errors.html
	}
	else {
		printf("Ошибка! дескриптор: NULL\n");
	}
	return curl;
	}
```

Файл «CheckServer.h»

```
#pragma once
#include <curl/curl.h>

CURL* checkServer();
	void getResponseData(char* data, size_t size, size_t n);
```

Файл «Configuration.h»

```
#pragma once
/*Файл настройки программы
*/
#include <locale.h>
#define use_UTF_8 setlocale(LC_ALL, "ru_RU.UTF-8")
#define use_Unicode setlocale(LC_ALL, "ru")
#define url "http://checkers24.ru/ibook/"
#define manual "для управления используйте следующие команды:\n"
#define manualexCommand "Для выхода: "
#define manualMessangerCommand ", для отправки сообщения: "
#define manualMessangerGetCommand ", для проверки пришедших сообщений: "
#define sendMessageCommand "send"
#define getMessageCommand "get"
#define exitCommand "exit"
#define history "lookhistory"
#define gethistory "Моя исторя сообщений:\n"
	#define manualhistory ", для получения истории сообщений: "
```

Файл «Messanger.cpp»

```
#define _CRT_SECURE_NO_WARNINGS
#include "Configuration.h"
```

```
#include <curl/curl.h>
#include "CheckServer.h"
#include "ibook.h"
#include "FormatJSON_String.h"
#include "File_manager.h"


void get_nickname(char* name) {
		// функция для ввода телефона
	printf("Введите имя: ");
	scanf("%s", name);
}

bool send_message_command(char* command) {
	// проверка ввода команд
	char ifCommand[] = sendMessageCommand;
	return strcmp(ifCommand, command);
}

bool get_message_command(char* command) {
	// проверка ввода команд
	char ifCommand[] = getMessageCommand;
	return strcmp(ifCommand, command);
}

bool get_history(char* command) {
		// проверка ввода команд
	char ifCommand[] = history;
	return strcmp(ifCommand, command);
}

void sendMessage(CURL* curl, char* name) {
	// отправка JSON стрроки (из send_message_JSON()) на сервер
	curl_easy_setopt(curl, CURLOPT_POSTFIELDS, send_message_JSON(name));
	curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, getResponseData);
	CURLcode res = curl_easy_perform(curl);
	expectationCommand(curl, name);
		// вызов функции из ibook.cpp для нового запуска мессенджера
}

void getMessage(CURL* curl, char* name) {
	// отправка JSON стрроки (из get_message_JSON()) на сервер
	curl_easy_setopt(curl, CURLOPT_POSTFIELDS, get_message_JSON(name));
	curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, getResponseData);
	CURLcode res = curl_easy_perform(curl);
		// вызов функции из ibook.cpp для нового запуска мессенджера
	expectationCommand(curl, name);
	}
```

Файл «Messanger.h»

```
#pragma once
#include <curl/curl.h>
void get_nickname(char* name);
bool send_message_command(char* command);
bool get_message_command(char* command);
bool get_history(char* command);
void sendMessage(CURL* curl, char* name);
void getMessage(CURL* curl, char* name);
```

Файл «FormatJSON_String.cpp»

```
#define _CRT_SECURE_NO_WARNINGS
#include "Messanger.h"
#include <string.h>
#include <stdio.h>
#include "File_manager.h"

extern "C" char* addStr(char* a, char* b, char* c);
	// функция сложения 3-х строк из .asm файла

char* send_message_JSON(char* name) {
	// функция формирует json строку, из которой сервер сохранит сообщение (поле
version)

	char recipient[13];
			// получатель
	char message[100];
			// сообщение
	char* str = new char[100];
			// json строка

	get_nickname(recipient);
		// получение номера отправителя
	scanf("%s", &message);
			// получение сообщения
	save_message(recipient, message);
		// сохранение сообщения

	str = addStr(
		(char*)"{\n\"request\": {\n\"command\": \" ",
		name, (char*)"\",\n\"phone\": \""
	);
				// сложение полученных строк пример: {"request": {
"command": " +79270969849","phone" : "

	/*Костыль!!!*/
	for (int i = 41 + strlen(name); i < 100; i++) {
	//при повторном вызове expectationCommand() строка почему-то получается с
мусором, форматирование строки
		str[i] = 0;
	}

	str = addStr(
		addStr(str, recipient, (char*)"\"\n},\n\"version\": \""),
	//сложение строк пример полученной строки: {"request": {"command": "
+79270969849","phone" : "+79270969879"},"version" : "hello!"}
		message, (char*)"\"\n}");
	return str;
}

char* get_message_JSON(char* name) {
	char* str = new char[100];

	str = addStr(
		(char*)"{\n\"request\": {\n\"command\": \"getMessage\",\n\"phone\":
\"",
		name,
		(char*)"\"\n},\n\"version\": \"\"\n}");
	/*Костыль!!!*/
	for (int i = 70 + strlen(name); i < 100; i++) {
		str[i] = 0;
	}
	return str;
	}
```

Файл «FormatJSON_String.h»

```
#pragma once
char* send_message_JSON(char* name);
	char* get_message_JSON(char* name);
```

Файл «addStr.asm»

```
includelib legacy_stdio_definitions.lib
.386
.model flat,stdcall
extern printf:near
extern scanf:near

;макрос в котором реализовано вычисление длины строк
len macro str:REQ
    cld
    mov edi,str
    mov esi,edi
    mov ecx, 0ffffffffh;
    xor al, al
    repne scasb
    sub edi,esi
    dec edi
    mov ecx, edi
endm

;вывод длин строк
printLen macro
    push ecx
    push edi
    push offset formatOutput
    call printf
    add esp, 4
    pop ecx
endm

.data
formatOutput db "%d", 10, 0
result db "
"
.code
public addStr
addStr proc C, stroka1:DWORD, stroka2:DWORD, stroka3:DWORD         ;процедура
принимает 3 строки и возвращает их склеенный результат

    mov ecx, 0
      len stroka1                                        ;вычисление длины первой
строки
    ;printLen                                          ;вывод длины (для отладки)
    mov edi,stroka1
    lea esi, result

addstr1:                                               ;запись данных из места в
оперативной памяти по адресу edi в esi
    mov al, byte ptr[edi]
    mov [esi],al
    inc edi
    inc esi                                            ;инкриминтирование для
следующего символа
loop addstr1
    push esi
    len stroka2                                        ;вычисление длины второй строки
```

```
   ; printLen                                          ;вывод длины (для отладки)
    mov edi,stroka2
    pop esi
addstr2:                                               ;запись данных из места в
оперативной памяти по адресу edi в esi
    mov al, byte ptr[edi]
    mov [esi],al
    inc edi
    inc esi                                            ;инкриминтирование для
следующего символа
loop addstr2
    push esi
    len stroka3                                        ;вычисление длины третьей строки
   ; printLen                                          ;вывод длины (для отладки)
    mov edi,stroka3
    pop esi
addstr3:                                               ;запись данных из места в
оперативной памяти по адресу edi в esi
    mov al, byte ptr[edi]
    mov [esi],al
    inc edi
    inc esi                                            ;инкриминтирование для
следующего символа
loop addstr3

    mov eax, offset result
      ret
addStr endp
      end
```

Файл «File_manager.cpp»

```
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
#include <Windows.h>
#include "Configuration.h"

extern "C" char* addStr(char* a, char* b, char* c);
      // функция сложения 3-х строк из .asm файла

void save_message(char* nameFile, char* message) {

      char fileName[13]{};

      for (int i = 0; i < 12; i++) {
            fileName[i] = nameFile[i];
      }

      nameFile = addStr(fileName, (char*)".", (char*)"txt");
      // получение имени файла
      /*Костыль!!!*/
      for (int i = strlen(fileName) + 4; i < 20; i++) {
      // при повторном вызове expectationCommand() строка почему-то получается с
мусором, форматирование строки
            nameFile[i] = 0;
      }
      FILE* file;
      file = fopen(nameFile, "a");
            // открытие файла
      fprintf(file, "%s\n", message);
            // запись
      fclose(file);
}
```

```
void readMessages(char* file) {
		//чтение сообщений
	FILE* fp;
	char string[1000] = "";
	if ((fp = fopen(file, "r")) == NULL)
	{
		printf("Не удалось открыть файл\n");
	}
	bool otherMessage = 0;
	while (!feof(fp)) {
		if (otherMessage) {
			// сообщения отправителя
			otherMessage = 0;
			printf("                                        %s\n", string);
		}
		else {
			if (string[0] == '+') {
				otherMessage = 1;
				// обнаружение сообщений отправителя
				printf("%s", string);
				// имя отправителя
			}
			else {
				printf("%s\n", string);
				// мои сообщения
			}
		}
		fscanf(fp, "%s[\n]", &string);
	}
	printf("\n");
}

void get_list_file() {
			// получение списка файлов
	const char fileTypes[] = "*.txt";

	WIN32_FIND_DATAA FindFileData;
	HANDLE hf = FindFirstFileA(fileTypes, &FindFileData);

	use_Unicode;
	printf(gethistory);
	if (hf != INVALID_HANDLE_VALUE) {
		do {
			printf("%s  ", FindFileData.cFileName);
		} while (FindNextFileA(hf, &FindFileData));
		FindClose(hf);
	}
	printf("\nВведите интересуемое имя чата или cancel - для отмены:\n");
	char file[17];
	scanf("%s", file);
	if (strcmp(file, (char*)"cancel")) {
		readMessages(file);
	}
}
```

Файл «File_manager.h»

```
#pragma once
void save_message(char* nameFile, char* message);
	void get_list_file();
```