

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский технологический
университет «МИСиС»

Институт информационных технологий и компьютерных наук
Кафедра Инженерной Кибернетики

КУРСОВАЯ РАБОТА

по дисциплине «Технологии программирования»

на тему:

«Консольный архиватор»

Разработал:

студент группы БПМ-19-2

Котелевский Артём Алексеевич

Проверил:

доцент кафедры ИК, к.т.н

Полевой Д.В.

Москва, 2020

СОДЕРЖАНИЕ

1	Описание задачи	3
1.1	Общие положения	3
1.2	Назначение разработки	3
1.3	Требования к программе	3
2	Пользовательское описание	4
2.1	Общие положения	4
2.2	Пользовательский интерфейс	4
3	Техническое описание.....	6
3.1	Классы и функции	6
3.1.1	Класс archivator_info.....	6
3.1.2	Класс command_line	7
3.1.3	Класс file_header	8
4	Сборка.....	9
5.	Тесты.....	10
5.1	Проверка работоспособности.....	10
5.2	Проверка создания архива	10
5.3	Проверка размера файла, заархивированного в 5.2	10
5.4	Проверка правильности разархивирования.....	10
ПРИЛОЖЕНИЕ А Проверка работоспособности на реальном примере.....		11

1 Описание задачи

1.1 Общие положения

Консольное приложение «Архиватор»(Далее - КП) включает в себя возможность использовать различные алгоритмы шифрования данных(в данной работе XOR шифрование).

КП предназначено для объединения файлов в архив, добавление новых файлов по мере их поступления, удаления файлов и разархивации всех файлов.

1.2 Назначение разработки

КП реализует набор классов, обеспечивающих следующие функции:

- Считывание данных из файла;
- Шифрование содержимого файла;
- Добавление и удаление файлов из архива;
- Получение списка файлов в архиве
- Разархивация файлов.

1.3 Требования к программе

Библиотека должна быть разработана в виде библиотеки классов на языке C++. Стандарт языка C++ не менее ISO C++17.

Средство сборки библиотеки: CMake (версии не менее 3.18), среда разработки: Microsoft Visual Studio 2019.

2 Пользовательское описание

2.1 Общие положения

КП реализует функционал архиватора для работы с файлами. Программа имеет 4 команды:

- 1) “-a” - добавление файлов в уже существующий архив или создание такого при его отсутствии.
- 2) “-d” – удаление файлов из уже существующего архива
- 3) “-l” – вывод списка файлов в архиве
- 4) “-x” – разархивация файлов из архива

Пример работы программы:

```
-a file.arch testdata.sql script+insert.sql tmp.txt  
-d file.arch testdata.sql  
-l file.arch  
-x file.arch
```

Команды передаются в КП посредством Command arguments в настройках Debugger-а, либо при переходе в директорию с исполняемым файлом. Первым параметром всегда вводится команда, далее идёт имя архива, после этого в зависимости от команды могут быть переданы имена файлов.

2.2 Пользовательский интерфейс

Если пользователь не знает как работают команды, либо введена неправильная команда будет выведен список команд с примерами:

```
List of commands:  
#-----#  
-a to add files to archive <-a tmp.arch tmp.txt> to add tmp.txt file into tmp.arch archive  
-----#  
-x to extract all files from archive <-x tmp.arch>  
-----#  
-d to delete files from archive <-d tmp.arch tmp.txt> to delete tmp.txt from archive tmp.arch!  
-----#  
-l to list all files in archive <-l tmp.arch>  
-----#  
#  
D:\DISK_D\Artem_new\C++\Archivator\02\bin.dbg\Archivator.exe (process 12316) exited with code 0.  
Press any key to close this window . . .  
-
```

При добавлении файлов(“-a”) в архив пользователь увидит информацию о файлах в архиве (их имена, размер и первый байт записи файла в архиве)

```
Add File_paths:
testdata.sql
script+insert.sql
tmp.txt
[=====] 100 %

testdata.sql
first_byte : 112
00 00 00 00 00 00 00 70
file_size : 2491
00 00 00 00 00 00 09 BB

script+insert.sql
first_byte : 2604
00 00 00 00 00 00 0A 2C
file_size : 5052
00 00 00 00 00 00 13 BC

tmp.txt
first_byte : 7657
00 00 00 00 00 00 1D E9
file_size : 344
00 00 00 00 00 00 01 58
```

При удалении файла из архива(“-d”) пользователь получит имена, размер и первый байт записи файла в архиве.

```
Delete File_paths:
testdata.sql
[=====] 100 %
file.arch

script+insert.sql
first_byte : 78
00 00 00 00 00 00 00 4E
file_size : 5052
00 00 00 00 00 00 13 BC

tmp.txt
first_byte : 5131
00 00 00 00 00 00 14 0B
file_size : 344
00 00 00 00 00 00 01 58

Written 5052 bytes out of 5052

Written 344 bytes out of 344
```

При вызове команды “-l” пользователь получит список файлов в архиве, а также их размер

```
List File_paths:
[=====] 100 %
File number 1 name: script+insert.sql
Size in bytes: 5052

File number 2 name: tmp.txt
Size in bytes: 344
```

При вызове команды “-x” пользователь получит разархивированные файлы в папке с проектом, а также сообщение о статусе разархивации.

```
Extract File_paths:
[=====] 100 %

Written 5052 bytes out of 5052

Written 344 bytes out of 344

Sucessfully unpacked files!
```

3 Техническое описание

3.1 Классы и функции

Объявление классов находится в файле «Archiver.h». Структура файла приведена в таблице 1.

Таблица 1 – Структура файла «Archiver.h»

Наименование	Описание
command_line	Класс работы с командной строкой
file_header	Класс работы с файлом для получения информации о нём
archivator_info	Класс работы с архивированным файлом

3.1.1 Класс archivator_info

Открытые члены

- void **get_file_info** (const std::string &file_name, **file_header** &fh)
получение информации о файле, размер, имя
- std::string **get_file_name** (std::string const &path)
метод получения имени из пути файла
- std::ifstream::pos_type **get_file_size** (const std::string &filename)
получает размер файла
- void **construct_header** (std::vector< std::string > &file_paths, std::vector< **file_header** > &headers, bool flag)
принимает команды
- void **pack** (const **command_line** &cl)
архивация файлов
- void **unpack** (const **command_line** &cl)
разархивация файлов
- void **delete_file** (const **command_line** &cl)
удаление файлов
- void **list** (const **command_line** &cl)
список файлов в архиве
- void **write_header** (std::ofstream &out, const std::vector< **file_header** > &headers)
метод архивации файла

- void **write_file** (std::ofstream &out, std::string file_name)
метод записи из файла в архив
- void **get_header_info** (std::ifstream &in, std::vector< **file_header** > &files)
получение информации из заголовка
- void **rewrite_files** (std::ifstream &in, std::ofstream &out, **file_header** file, bool action)
- void **int64ToChar** (unsigned char mesg[], int64_t num)
перевод из int64 к символьному типу
- void **XOR_cipher_decipher** (char &original_char)
метод XOR шифрования
- int64_t **charTo64bitNum** (unsigned char a[])
перевод из символьного типа к int64

Открытые статические члены

- static void **check_file** (std::string file_name)
метод проверки существования файла

Закрытые данные

- std::vector<file_header> file_info;
вектор, хранящий информацию о файлах
- int file_amount = 0;
кол-во файлов
- char signature[6] = "art25";
сигнатура архива

Объявления и описания членов классов находятся в файлах:

- Archiver.h
- Archiver.cpp

3.1.2 Класс **command_line**

Открытые члены

- **command_line** ()
конструктор
- command_t **get_command** ()

получает нужную команду

- `const std::string & get_archive_name () const`
получает имя архива
- `const std::vector< std::string > & get_file_names () const`
получает имена файлов
- `void set_file_names (std::vector< std::string > file_names)`
заполняет вектор именами файлов, для работы с ними
- `void set_archive_name (std::string archive_name)`
устанавливает имя архива
- `bool parse (int argc, char *argv[])`
- `void parse_file_names (int argc, char *argv[])`
считывает имена файлов
- `void print ()`
- `void print_command_list ()`
выводит список команд

Открытые статические члены

- `static void progress_bar ()`
выводит полосу загрузки

Закрытые члены

- `command_t` `m_command;`
какая команда выбрана
- `std::string` `m_archive_name;`
имя архива
- `std::vector<std::string>` `m_file_names;`
имена файлов

Объявления и описания членов классов находятся в файлах:

- Archiver.h
- Archiver.cpp

3.1.3 Класс file_header

Открытые члены

- `void set_name (const std::string &name)`

устанавливает имя файла

- `const std::string & get_name ()`
получает имя файла
- `void set_size (int64_t size)`
устанавливает размер файла
- `int64_t get_size ()`
получает размер файла
- `void set_first_byte (int64_t first_byte)`
записывает в хедер первый байт файла в архиве
- `int64_t get_first_byte ()`
получает из хедера первый байт файла

Закрытые данные

- `std::string m_name;`
имя файла
- `int64_t m_size;`
размер файла
- `int64_t m_first_byte;`
первый байт файла

Объявления и описания членов класса находятся в файле:

- `Archiver.h`

4 Сборка

1. Загрузить проект, размещенный в svn-хранилище по ссылке:
https://mysvn.ru/Artem_Kotelevsky/CourseWork/
2. В программе `cmake-gui.exe` указать путь до корневой папки проекта, также нужно указать компилятор.
3. После этого введите директорию, в которой вы собираетесь построить проект в поле `CMAKE_INSTALL_PREFIX`.
4. Теперь нужно построить проект.
5. Запускаем через `cmake` проект и выбираем `Archivator` как `Startup Project`

6. Передаем аргументы через Debug -> Archivator Properties -> Debugging -> Command Arguments

5 Тесты.

5.1 Проверка работоспособности программы.

1. Запустить программу с пустым списком аргументов.
2. Получить список команд.
3. Заккрыть диалоговое окно.

5.2 Проверка создания архива.

1. Запустить программу с командой -а <название вашего архива> <название файла, который вы добавите в папку исполняемого проекта>
2. Заккрыть диалоговое окно
3. Найти в папке с исполняемым проектом файл с названием вашего архива

5.3 Проверка размера файла, заархивированного в 5.2.

1. Запустить программу с командой -l <название вашего архива>
2. В диалоговом окне убедиться в правильности размера сохраненного вами файла

5.4 Проверка правильности разархивирования.

1. Запустить программу с командой -x <название вашего архива>
2. Проверить файл на правильность записи

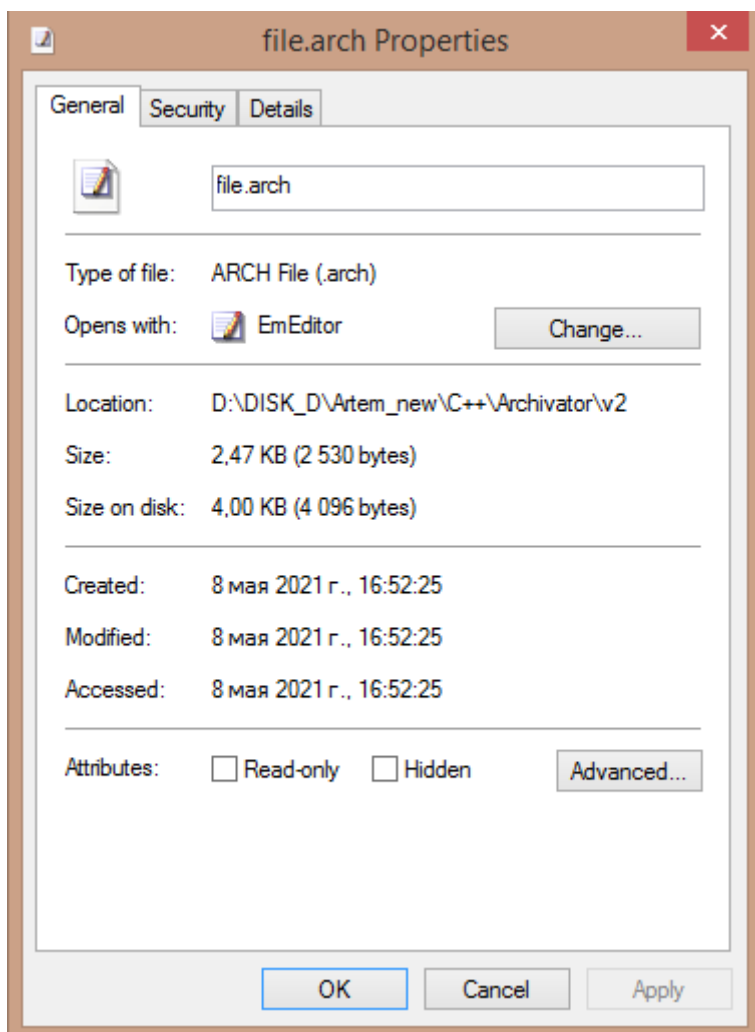
ПРИЛОЖЕНИЕ А

Проверка работоспособности на реальном примере

- 1) Вводим команду -a file.arch testdata.sql где file.arch имя архива, а testdata.sql имя добавляемого в архив файла

```
Add File_paths:
testdata.sql
[=====] 100 %
-----
testdata.sql
first_byte : 40
00 00 00 00 00 00 00 28
file_size : 2491
00 00 00 00 00 00 09 BB
D:\DISK_D\Artem_new\C++\Archivator\v2\Debug\Archivator.exe (process 9128) exited with code 0.
Press any key to close this window . . .
```

- 2) Видим, что размер нашего архива 2530. Запись файла начиналась с 40 байта и размер файла 2491 => архив создан верно



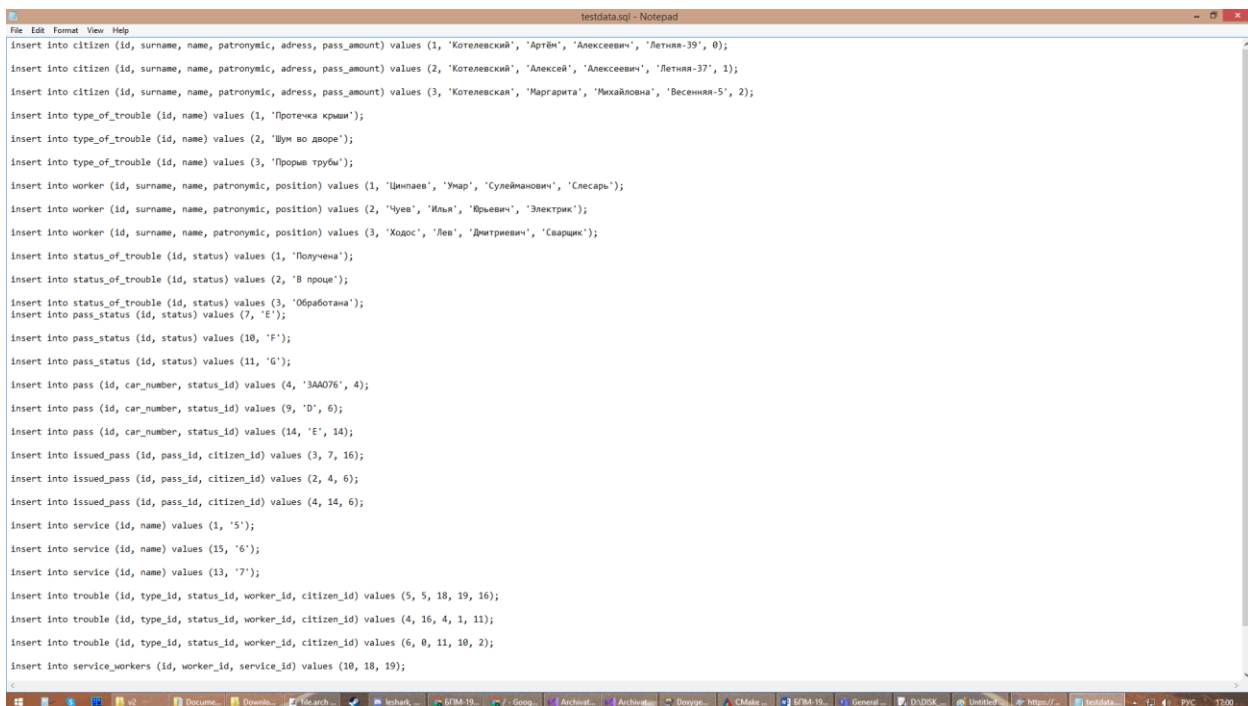
3) Введем команду -l file.arch чтобы проверить правильность записи

```
List File_paths:
[=====] 100 %
File number 1 name: testdata.sql
Size in bytes: 2491
-----
D:\DISK_D\Artem_new\C++\Archivator\v2\Debug\Archivator.exe <process 9732> exited with code 0.
Press any key to close this window . . .
```

4) Введем команду -x file.arch чтобы разархивировать созданный архив и проверим правильность разархивации

```
Extract File_paths:
[=====] 100 %
-----
Written 2491 bytes out of 2491
-----
Successfully unpacked files!
D:\DISK_D\Artem_new\C++\Archivator\v2\Debug\Archivator.exe <process 9940> exited with code 0.
Press any key to close this window . . .
```

Файл до:



```
File Edit Format View Help
testdata.sql - Notepad
Insert into citizen (id, surname, name, patronymic, address, pass_amount) values (1, 'Котелевский', 'Артём', 'Алексеевич', 'Летняя-39', 0);
Insert into citizen (id, surname, name, patronymic, address, pass_amount) values (2, 'Котелевский', 'Алексей', 'Алексеевич', 'Летняя-37', 1);
Insert into citizen (id, surname, name, patronymic, address, pass_amount) values (3, 'Котелевская', 'Маргарита', 'Михайловна', 'Весенняя-5', 2);
Insert into type_of_trouble (id, name) values (1, 'Протечка крыши');
Insert into type_of_trouble (id, name) values (2, 'Шум во дворе');
Insert into type_of_trouble (id, name) values (3, 'Прорыв трубы');
Insert into worker (id, surname, name, patronymic, position) values (1, 'Щипаев', 'Умар', 'Сулейманович', 'Слесарь');
Insert into worker (id, surname, name, patronymic, position) values (2, 'Чуев', 'Илья', 'Васильевич', 'Электрик');
Insert into worker (id, surname, name, patronymic, position) values (3, 'Ходос', 'Лев', 'Дмитриевич', 'Сварщик');
Insert into status_of_trouble (id, status) values (1, 'Получена');
Insert into status_of_trouble (id, status) values (2, 'В проuce');
Insert into status_of_trouble (id, status) values (3, 'Обработана');
Insert into pass_status (id, status) values (7, 'E');
Insert into pass_status (id, status) values (10, 'F');
Insert into pass_status (id, status) values (11, 'G');
Insert into pass (id, car_number, status_id) values (4, '3AA076', 4);
Insert into pass (id, car_number, status_id) values (9, 'D', 6);
Insert into pass (id, car_number, status_id) values (14, 'E', 14);
Insert into issued_pass (id, pass_id, citizen_id) values (3, 7, 16);
Insert into issued_pass (id, pass_id, citizen_id) values (2, 4, 6);
Insert into issued_pass (id, pass_id, citizen_id) values (4, 14, 6);
Insert into service (id, name) values (1, '5');
Insert into service (id, name) values (15, '6');
Insert into service (id, name) values (13, '7');
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (5, 5, 18, 19, 16);
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (4, 16, 4, 1, 11);
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (6, 0, 11, 10, 2);
Insert into service_workers (id, worker_id, service_id) values (10, 18, 19);
```

Файл после:

```
testdata.sql - Notepad
File Edit Format View Help
Insert into citizen (id, surname, name, patronymic, adress, pass_amount) values (1, 'Котелевский', 'Артём', 'Алексеевич', 'Летняя-39', 0);
Insert into citizen (id, surname, name, patronymic, adress, pass_amount) values (2, 'Котелевский', 'Алексей', 'Алексеевич', 'Летняя-37', 1);
Insert into citizen (id, surname, name, patronymic, adress, pass_amount) values (3, 'Котелевская', 'Маргарита', 'Михайловна', 'Весенняя-5', 2);
Insert into type_of_trouble (id, name) values (1, 'Протечка крыши');
Insert into type_of_trouble (id, name) values (2, 'Шум во дворе');
Insert into type_of_trouble (id, name) values (3, 'Прорыв трубы');
Insert into worker (id, surname, name, patronymic, position) values (1, 'Цинпаев', 'Умар', 'Сулейманович', 'Слесарь');
Insert into worker (id, surname, name, patronymic, position) values (2, 'Чуев', 'Илья', 'Врьевич', 'Электрик');
Insert into worker (id, surname, name, patronymic, position) values (3, 'Ходос', 'Лев', 'Дмитриевич', 'Сварщик');
Insert into status_of_trouble (id, status) values (1, 'Получена');
Insert into status_of_trouble (id, status) values (2, 'В проце');
Insert into status_of_trouble (id, status) values (3, 'Обработана');
Insert into pass_status (id, status) values (7, 'E');
Insert into pass_status (id, status) values (10, 'F');
Insert into pass_status (id, status) values (11, 'G');
Insert into pass (id, car_number, status_id) values (4, '3AA076', 4);
Insert into pass (id, car_number, status_id) values (9, 'D', 6);
Insert into pass (id, car_number, status_id) values (14, 'E', 14);
Insert into issued_pass (id, pass_id, citizen_id) values (3, 7, 16);
Insert into issued_pass (id, pass_id, citizen_id) values (2, 4, 6);
Insert into issued_pass (id, pass_id, citizen_id) values (4, 14, 6);
Insert into service (id, name) values (1, '5');
Insert into service (id, name) values (15, '6');
Insert into service (id, name) values (13, '7');
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (5, 5, 18, 19, 16);
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (4, 16, 4, 1, 11);
Insert into trouble (id, type_id, status_id, worker_id, citizen_id) values (6, 0, 11, 10, 2);
Insert into service_workers (id, worker_id, service_id) values (10, 18, 19);
```

Файл разархивирован правильно, что означает правильность работы программы.