

Министерство образования Республики Беларусь  
Учреждение Образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Военный факультет

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Архиватор файлов и директорий (Windows)

БГУИР КП 1-40 02 01 002 ПЗ

Студент: группы 830501,

Глинка.А.В.

Руководитель: ассистент каф. ЭВМ,

Желтко Ю.Ю.

Минск 2020

Учреждение образования  
«Белорусский государственный университет информатики  
и радиоэлектроники»

Военный факультет

УТВЕРЖДАЮ  
Заведующий кафедрой

\_\_\_\_\_  
(подпись)

\_\_\_\_\_  
20 г.

ЗАДАНИЕ  
по курсовому проектированию

Студенту Глинке Андрею Владимировичу.

1. Тема проекта Архиватор файлов и директорий (Windows).
2. Срок сдачи студентом законченного проекта 20 мая 2020 г.
3. Исходные данные к проекту Операционная система – Windows, Язык программирования – C++.
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке) Введение. 2. Обзор литературы 3. Системное проектирование. 4. Функциональное проектирование 5. Разработка программных модулей. 6. Программа и методика испытаний 7. Руководства пользователя. Заключение. Список использованных источников
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) 1. Схема структурная. 2. Диаграмма классов
6. Консультант по проекту Желтко Ю.Ю.
7. Дата выдачи задания 22 февраля 2020 г.
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):  
разделы 1,2 к 1 марта 2020 г. – 20 %;  
разделы 3,4 к 1 апреля 2020 г. – 30 %;  
разделы 5,6,7 к 1 мая 2020 г. – 30 %;  
оформление пояснительной записки и графического материала к 20 мая 2020 г. 20 %  
Защита курсового проекта с 01 июня 2020 г. по 10 июня 2020 г.

РУКОВОДИТЕЛЬ \_\_\_\_\_ Ю.Ю. Желтко  
(подпись)

Задание принял к исполнению \_\_\_\_\_ А.В.Глинка  
(дата и подпись студента)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1.ОЗОР ЛИТЕРАТУРЫ .....	6
1.1.Основные термины .....	6
1.2.Обзор аналогов .....	6
2.СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ .....	8
2.1.Модуль управления приложением .....	8
2.2.Модуль пользовательского интерфейса .....	8
3.ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....	9
3.1.Выбор языка.....	9
3.2.Выбор среды разработки архиватора.....	9
3.3.Описание функционирования программы .....	9
4.РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	10
4.1.Схема алгоритма .....	10
4.2.Алгоритм по шагам.....	10
5.ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ.....	11
5.1.Пользовательский ввод.....	11
5.2.Тестирование архивации данных .....	11
6.РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	12
6.1.Системные требования .....	12
6.2.Использование приложения.....	13
ЗАКЛЮЧЕНИЕ .....	14
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	15
ПРИЛОЖЕНИЕ А .....	16
ПРИЛОЖЕНИЕ Б.....	18
ПРИЛОЖЕНИЕ В .....	20
ПРИЛОЖЕНИЕ Г.....	36

## ВВЕДЕНИЕ

Архиватор — программа, предназначенная для упаковки без потерь одного и более файлов в единый файл-архив или в серию архивов для удобства переноса и/или хранения данных. Распаковка архивов выполняется с помощью того же архиватора либо посредством сторонних совместимых утилит. Большинство современных архиваторов также выполняют сжатие упаковываемых в архив данных.

Простейшие архиваторы просто последовательно объединяют (упаковывают) содержимое файлов в архив, который, помимо файловых данных, содержит информацию об именах и размерах исходных файлов, чтобы можно было точно восстанавливать файлы в их первоизданном виде. Поэтому большинство архиваторов также сохраняет метаданные файлов, предоставляемые операционной системой, такие как время создания и права доступа. Такую функциональность реализует tar — стандартный архиватор систем типа UNIX. Если необходимо уменьшить размер tar-архива, к нему применяют сжатие без потерь программами gzip, bzip2 и т. д. Большинство современных прикладных архиваторов использует сжатие при работе с настройками по умолчанию.

Многие архиваторы позволяют указывать дополнительные параметры, наиболее важные из которых влияют на степень и скорость сжатия. Эти характеристики — обратно зависимые величины. То есть, чем выше скорость упаковки, тем меньше степень сжатия, и наоборот. Что касается скорости распаковки, то в большинстве современных архиваторов применяются так называемые асимметричные алгоритмы сжатия, при которых скорость (и степень) упаковки практически не влияет на скорость распаковки, которая обычно гораздо выше.

Большинство архиваторов имеет функцию проверки целостности хранящихся в архиве данных. Для этого в архив при добавлении туда файлов вносится информация об их контрольных суммах. При распаковке (или тестировании) архива обязательно вычисляется контрольная сумма каждого извлекаемого файла, и, если она не совпадает с суммой, хранящейся в архиве, то выводится сообщение об ошибке. Таким образом, архиваторы предоставляют очень важную возможность, о которой многие даже не задумываются: гарантию целостности данных. Кроме того, некоторые архиваторы (например RAR) имеют функции защиты архивов от физических повреждений или даже полной утери отдельных томов многотомных архивов, благодаря чему архив можно рассматривать не только как средство для хранения данных, но и для их восстановления в исходном виде в случае повреждений.

Некоторые архиваторы позволяют создавать так называемые многотомные архивы, то есть архивы, состоящие из нескольких частей указанного или разного размера. Такие архивы удобно применять для переноса больших объёмов данных на носителях меньшего размера

(например) на флэшках или оптических дисках) и обмена данными через Интернет, когда вместо одного огромного архива практичнее передать несколько файлов меньшего размера. В разных архиваторах многотомность реализована по-разному. Например, в форматах ZIP и 7Z тома — это, по сути, просто разделённый на несколько частей исходный архив, что накладывает определённые ограничения на их использование, тогда как тома многотомных архивов RAR представляют собой практически полноценные архивы.

У ряда архиваторов имеется дополнительная функция создания самораспаковывающихся (SFX) архивов. Такие архивы представляют собой исполняемые файлы, для распаковки которых не требуются никакие другие программы — нужно просто запустить SFX-архив, и он сам извлечёт все содержащиеся в нём данные. Это удобно, когда нужно передать архив кому-то ещё, но нет уверенности, что у него окажется соответствующий архиватор. В действительности SFX-архив — это обычный архив, к которому прикреплён исполняемый модуль распаковки, поэтому SFX-архивы можно обрабатывать внешним архиватором как обычные архивы (например, из-за опасения, что исполняемый модуль может быть заражён вирусом). SFX-архивы, создаваемые некоторыми архиваторами, могут быть многотомными, в этом случае первый том имеет исполняемый формат файла, а все последующие — стандартный для томов.

Сжатие файлов зависит от используемого архиватором алгоритма и от характера содержащихся в файлах данных. Текстовые файлы поддаются сжатию очень хорошо, бинарные файлы, как правило, хуже, а файлы, содержимое которых уже максимально уплотнено (например аудио- и видеофайлы, а также программы-инсталляторы), — и вовсе не сжимаются.

Если в архив упаковывается много файлов, имеющих похожую структуру данных, то некоторые архиваторы (например RAR и 7-Zip) позволяют получать архивы существенно меньшего размера благодаря использованию так называемого непрерывного (solid) сжатия. При таком сжатии все входящие файлы рассматриваются как один непрерывный поток данных, для которого используется один общий словарь, благодаря чему можно достичь очень высокой общей степени сжатия. При этом следует иметь в виду недостатки «непрерывных» архивов: их дольше обновлять; чтобы извлечь файл, находящийся в конце архива, нужно распаковать все файлы, идущие перед ним; в случае повреждения извлечь удастся только файлы, идущие в архиве перед повреждённым файлом.

# 1.ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Основные термины

Архиватор — программа, предназначенная для упаковки без потерь одного и более файлов в единый файл-архив или в серию архивов для удобства переноса и/или хранения данных. Распаковка архивов выполняется с помощью того же архиватора либо посредством сторонних совместимых утилит. Большинство современных архиваторов также выполняют сжатие упаковываемых в архив данных.

SFX-архив — это обычный архив, к которому прикреплен исполняемый модуль распаковки, поэтому SFX-архивы можно обрабатывать внешним архиватором как обычные архивы.

Сжатие данных (англ. data compression) — алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объема. Применяется для более рационального использования устройств хранения и передачи данных.

Коэффициент сжатия — основная характеристика алгоритма сжатия. Она определяется как отношение объема исходных несжатых данных к объему сжатых данных.

Архив — это файл, содержащий в себе один или несколько других файлов и/или папок, а также метаданные. Архивы используются для объединения множества любых файлов в единый файл-контейнер с целью удобства хранения и переноса информации или просто чтобы сжать данные. Для создания архивов и работы с ними используются программы-архиваторы.

В архивах может сохраняться структура папок, присутствовать служебная информация для обнаружения и исправления ошибок, комментарии и другая информация. В зависимости от формата архива, данные в нём могут шифроваться с помощью пароля.

## 1.2 Обзор аналогов

Простейшие архиваторы просто последовательно объединяют (упаковывают) содержимое файлов в архив, который, помимо файловых данных, содержит информацию об именах и размерах исходных файлов, чтобы можно было точно восстанавливать файлы в их первоизданном виде. Поэтому большинство архиваторов также сохраняет метаданные файлов, предоставляемые операционной системой, такие как время создания и права доступа.

Многие архиваторы позволяют указывать дополнительные параметры, наиболее важные из которых влияют на степень и скорость сжатия. Эти характеристики — обратно зависимые величины. То есть, чем выше скорость упаковки, тем меньше степень сжатия, и наоборот. Что касается скорости распаковки, то в большинстве современных архиваторов применяются так

называемые асимметричные алгоритмы сжатия, при которых скорость (и степень) упаковки практически не влияет на скорость распаковки, которая обычно гораздо выше.

Большинство архиваторов имеет функцию проверки целостности хранящихся в архиве данных. Для этого в архив при добавлении туда файлов вносится информация об их контрольных суммах. При распаковке (или тестировании) архива обязательно вычисляется контрольная сумма каждого извлекаемого файла, и, если она не совпадает с суммой, хранящейся в архиве, то выводится сообщение об ошибке. Таким образом, архиваторы предоставляют очень важную возможность, о которой многие даже не задумываются: гарантию целостности данных. Кроме того, некоторые архиваторы (например RAR) имеют функции защиты архивов от физических повреждений или даже полной утери отдельных томов многотомных архивов, благодаря чему архив можно рассматривать не только как средство для хранения данных, но и для их восстановления в исходном виде в случае повреждений.

У ряда архиваторов имеется дополнительная функция создания самораспаковывающихся (SFX) архивов. Такие архивы представляют собой исполняемые файлы, для распаковки которых не требуются никакие другие программы — нужно просто запустить SFX-архив, и он сам извлечёт все содержащиеся в нём данные. Это удобно, когда нужно передать архив кому-то ещё, но нет уверенности, что у него окажется соответствующий архиватор. В действительности SFX-архив — это обычный архив, к которому прикреплён исполняемый модуль распаковки, поэтому SFX-архивы можно обрабатывать внешним архиватором как обычные архивы (например, из-за опасения, что исполняемый модуль может быть заражён вирусом). SFX-архивы, создаваемые некоторыми архиваторами, могут быть многотомными, в этом случае первый том имеет исполняемый формат файла, а все последующие — стандартный для томов.

Наиболее популярные программные средства, предназначенные для архивирования файлов:

1. WinRAR.
2. 7-Zip.
3. Ark.
4. Zipeg.

## **2.СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

При разработке любой программы всегда стоит группировать отдельные ее элементы в функциональные блоки, что упростит понимание программы, а также повысит ее эффективность. Традиционно такое приложение можно поделить на два блока: управление приложением и пользовательский интерфейс.

### **2.1.Модуль управления приложением**

Данный модуль сосредотачивает в себе всю основную логику приложения - в данном случае работа с файлами, алгоритм архивации/разархивации, алгоритм создания двоичное лексикографически упорядоченное дерево для поиска, чтение байтов, а также вычисление степени сжатия исходного файла. Предположительно планируется добавить возможность копирования файлов, их удаление, функцию архивирования сразу нескольких файлов в один архив, создание самораспаковывающихся архивов(SFX) которые могли бы при запуске сами извлекать из себя содержащиеся в нем данные, а также разработку функции защиты данных от физических повреждений или же от полной утери отдельных томов многотомных архивов.

### **2.2.Модуль пользовательского интерфейса**

В данном модуле осуществляется вывод необходимой для пользователя информации на экран: выбор архивации/разархивации, выход из программы, показ исходной величины файла и полученного. Планируется добавить графический интерфейс для более удобного взаимодействия между пользователем и программой. Внедрение в него возможность выбора директорий и кнопок быстрого доступа.



### **3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

#### **3.1. Выбор языка**

C++ – компилируемый статически типизированный язык программирования общего назначения. Его самые главные преимущества – быстрота, мощность и относительная простота. Обратная полная поддержка языка C, его основателя, позволяет использовать инструменты, написанные много ранее его выпуска. Благодаря его положительным качествам и возможности непосредственной работы с памятью и поддержки ассемблерных вставок, он повсеместно используется для разработки системного ПО. Для данной курсовой работы это лучший выбор, так как основная нагрузка приходится именно на работу с памятью и её быструю обработку алгоритмами.

#### **3.2. Выбор среды для разработки архиватора**

Для разработки архиватора используются система: Windows. Для разработки используется её последняя версия Windows10. Наиболее распространенной средой разработки на C++ под Windows является Visual Studio ей я и буду пользоваться для написания данного курсового проекта.

#### **3.3. Описание функционирования программы**

Исходя из функционала данное программное обеспечение должно создавать и распаковывать архивы. Данный функционал реализован с помощью алгоритма LZSS. Для максимального упрощения и кроссплатформенности используется командная строка, основными аргументами которой являются путь к исходному файлу и функция его обработки – архивация или разархивация.

## **4.РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В этом разделе рассматривается схема алгоритма, а также описываются другой используемый в программе алгоритм.

### **4.1.Схема алгоритма**

Схема алгоритма добавления строки в двоичное дерево поиска представлена в Приложении Б.

### **4.2.Алгоритм по шагам**

Для алгоритма по шагам рассмотрена процедура декодирования сжатого файла:

- 1.Начало
- 2.Входные данные: input данный файл который мы архивируем, output архивированный файл
- 3.Читаем количество символов в файле
- 4.Если в файле присутствует один символ, то он кодируется 8 битами
- 5.И т.к. мы работаем с кольцевым буфером мы преобразуем индекс для его записи  $current\_pos = MODULO(current\_pos + 1)$
- 6.Если же в файле больше одного символа, то мы считываем в переменную `match_pos` количество битов, содержащих смещение совпадения, если их нет то заканчиваем процедуру
- 7.В переменную `match_len` записываем количество битов, содержащих длину совпадения
- 8.Выполняем тоже действие, как в пункте 5
- 9.Конец

## 5.ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

### 5.1.Пользовательский ввод

В программе производится проверка пользовательского ввода, если пользователь указывает несуществующий путь, то он получает уведомление об этом.

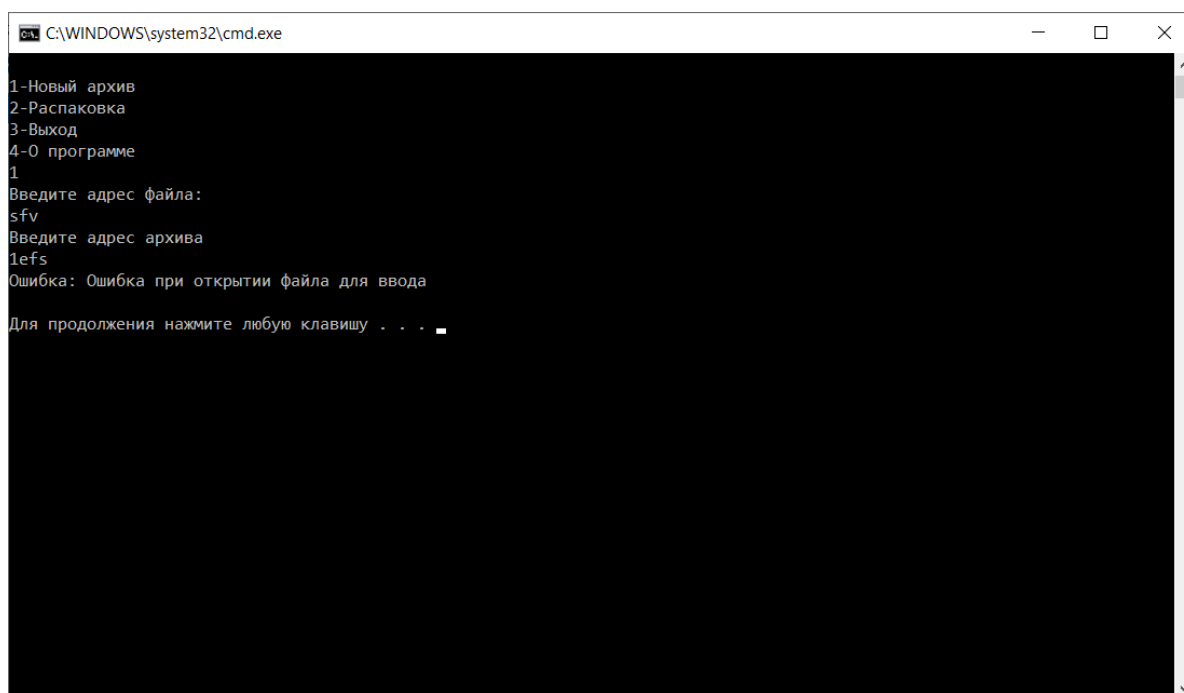
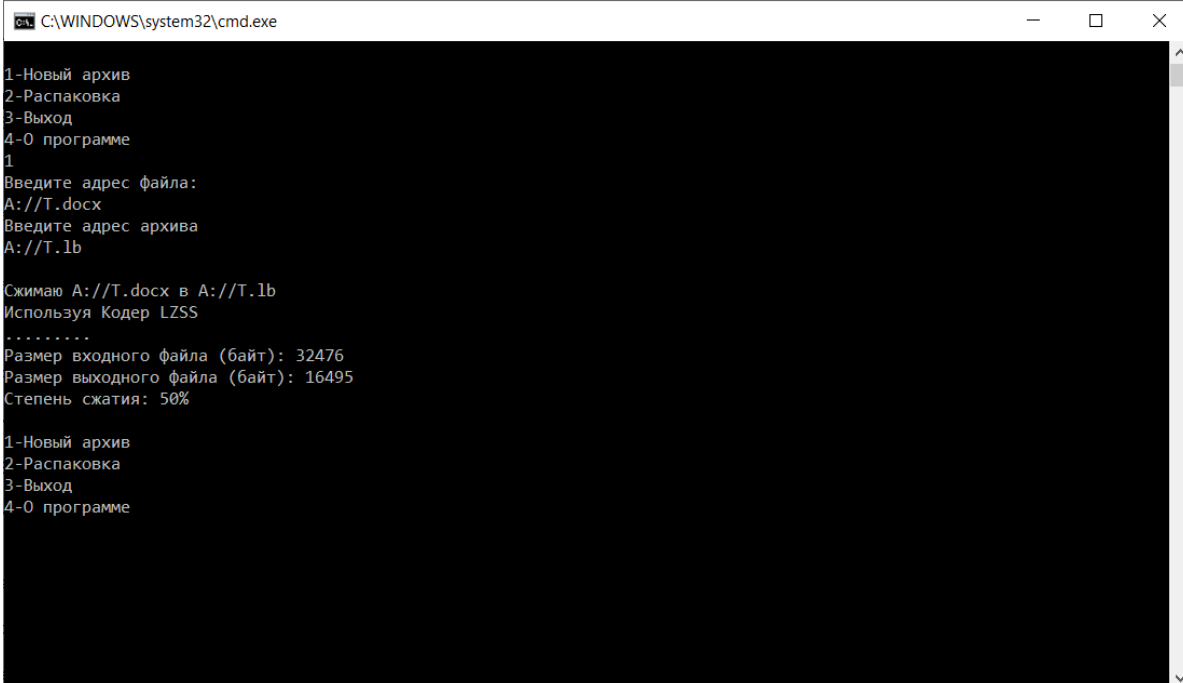


Рисунок 5.1 – Проверка на ввод адреса

## 5.2. Тестирование архивации данных

В ходе тестирования приложение архивировало и восстанавливало архивы в исходные данные без повреждений и потерь



```
C:\WINDOWS\system32\cmd.exe

1-Новый архив
2-Распаковка
3-Выход
4-0 программе
1
Введите адрес файла:
A://T.docx
Введите адрес архива
A://T.lb

Сжимаю A://T.docx в A://T.lb
Используя Кодер LZSS
.....
Размер входного файла (байт): 32476
Размер выходного файла (байт): 16495
Степень сжатия: 50%

1-Новый архив
2-Распаковка
3-Выход
4-0 программе
```

Рисунок 5.2 – Проверка на степень сжатия исходного файла

## **6.РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

### **6.1.Системные требования**

Системные требования – перечисление параметров компьютера, необходимых для того, чтобы на нем работало приложение, к которому собственно эти самые системные требования и написаны. Эти характеристики могут описывать требования как к компонентам, которыми может быть напичкан системный блок (тип процессора и его частота, модель видеокарты, объём оперативной памяти), так и к софту, установленному на компьютере (операционная система, поддержка и наличие какой-то программы или сторонних сервисов). Требования составляются разработчиком приложения.

Данный разработанный проект рассчитан даже для малопроизводительных компьютеров, о чем свидетельствуют нижеуказанные системные требования:

Минимальные системные требования:

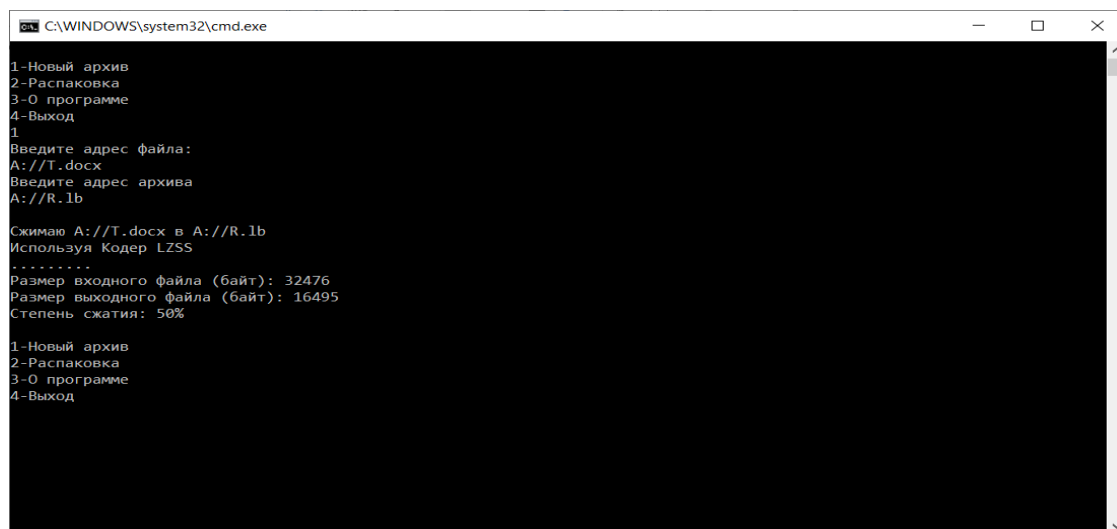
Операционная система: Windows 7  
Процессор: Pentium III или AMD Athlon  
Оперативная память: 512 Mb  
HDD: 32 Mb на жестком диске  
Видеокарта: с 64 МБ видеопамяти  
А также: Клавиатура

Рекомендуемые системные требования:

Операционная система: Windows 10  
Процессор: Intel Pentium III или AMD Athlon  
Оперативная память: 512 Mb  
HDD: 32 Mb на жестком диске  
Видеокарта: со 128 и более МБ видеопамяти  
А также: Клавиатура

## 6.2.Использование приложения

При запуске приложения выводится консоль, для архивирования файла необходимо выбрать первый пункт, а затем необходимо ввести адрес исходного файла с его расширением для архивации, потом директорию и расширение, в котором сохранится выходной архив.



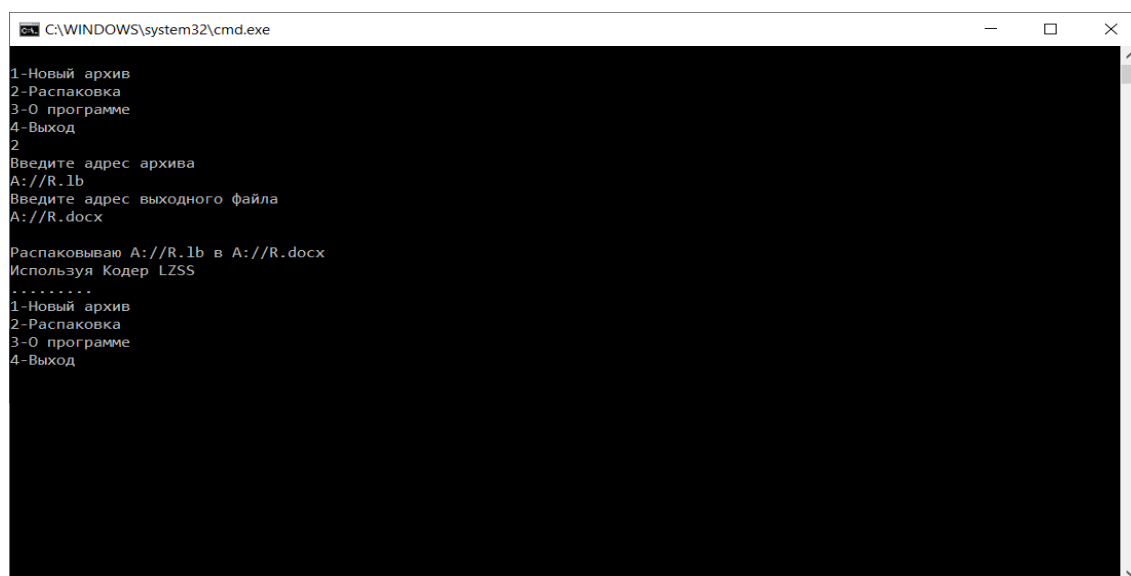
```
C:\WINDOWS\system32\cmd.exe
1-Новый архив
2-Распаковка
3-О программе
4-Выход
1
Введите адрес файла:
A://T.docx
Введите адрес архива
A://R.lb

Сжимаю A://T.docx в A://R.lb
Используя Кодер LZSS
.....
Размер входного файла (байт): 32476
Размер выходного файла (байт): 16495
Степень сжатия: 50%

1-Новый архив
2-Распаковка
3-О программе
4-Выход
```

Рисунок 6.1 – Алгоритм архивирования файла

Для того чтобы разархивировать файл необходимо выбрать второй пункт меню, ввести адрес архивированного файла и директорию куда он разархивируется.



```
C:\WINDOWS\system32\cmd.exe
1-Новый архив
2-Распаковка
3-О программе
4-Выход
2
Введите адрес архива
A://R.lb
Введите адрес выходного файла
A://R.docx

Распаковываю A://R.lb в A://R.docx
Используя Кодер LZSS
.....
1-Новый архив
2-Распаковка
3-О программе
4-Выход
```

Рисунок 6.2. – Алгоритм разархивирования файла

## ЗАКЛЮЧЕНИЕ

Архиватор — программа, предназначенная для упаковки без потерь одного и более файлов в единый файл-архив или в серию архивов для удобства переноса и/или хранения данных. Распаковка архивов выполняется с помощью того же архиватора либо посредством сторонних совместимых утилит. Большинство современных архиваторов также выполняют сжатие упаковываемых в архив данных.

В результате выполнения курсового проекта было разработано консольное приложение «Архиватор файлов» под операционную систему Windows на многофункциональном, компилируемом языке программирования общего назначения C++. Данный язык программирования был выбран в связи с тем, что он является одним из наиболее популярных и простых в освоении.

В программе удалось реализовать быстрый и простой способ архивирования файлов, а также разархивирование этого файла без потери данных.

Не реализована возможность закрытия архивирования сразу нескольких файлов, а также графический интерфейс для более понятного взаимодействия с программой, данная проблема связана с тем, что у автора данного проекта были недостаточно обширные знания в сфере архивирования файлов.

В дальнейшем планируется не только устранить данные проблемы, но и улучшить уже имеющиеся сильные стороны. Во-первых, ускорить архивирование “тяжелых файлов”. Во-вторых, уменьшить размер полученного файла.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1].habr.com/ - электронный веб-ресурс в формате коллективного блога посвященный сфере IT и программированию
- [2].cyberforum.ru/ - форум, посвященный решению различных вопросов по программированию
- [3].ravesli.com/ - образовательный портал, на который выкладываются переводы полезных статей по C++
- [4].Луцик Ю. А. «Объектно-ориентированное программирование на языке C++: учеб. Пособие» Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008г.
- [5].Теоретические основы алгоритмизации и программирования [Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/4693947/> – Дата доступа 19.10.2019.
- [6]Луцик, Ю. А. Объектно-ориентированное программирование на языке C++: учеб. пособие по курсу «Объектно-ориентированное программирование» для студ. спец. «Вычислительные машины, системы и сети» всех форм обуч. / Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова. – Минск: БГУИР, 2003. – 203 с.:ил.



**ПРИЛОЖЕНИЕ А**  
*(обязательное)*  
Схема структурная

**ПРИЛОЖЕНИЕ Б**  
*(обязательное)*  
Схема алгоритма

## ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <clocale>

typedef unsigned char uchar;
typedef unsigned long ulong;
using namespace std;

typedef struct bfile
{
    FILE *file;
    uchar mask;
    int rack;
    int pacifier_counter;
}
BFILE;

#define PACIFIER_COUNT 2047

BFILE *OpenInputBFile(char *name);
BFILE *OpenOutputBFile(char *name);
void WriteBit(BFILE *bfile, int bit);
void WriteBits(BFILE *bfile, ulong code, int count);
int ReadBit(BFILE *bfile);
ulong ReadBits(BFILE *bfile, int bit_count);
void CloseInputBFile(BFILE *bfile);
void CloseOutputBFile(BFILE *bfile);

void CompressFile(FILE *input, BFILE *output);
void ExpandFile(BFILE *input, FILE *output);
void usage_exit(char *prog_name);
void print_ratios(char *input, char *output);
long file_size(char *name);

void InitTree(int r);
void ContractNode(int old_node, int new_node);
void ReplaceNode(int old_node, int new_node);
int FindNextNode(int node);
void DeleteString(int p);
int AddString(int new_node, int *match_position);
```

```

#define INDEX_BITS 12
#define LENGTH_BITS 4
#define WINDOW_SIZE ( 1 << INDEX_BITS)
#define RAW_LOOK_AHEAD_SIZE ( 1 << LENGTH_BITS)
#define BREAK_EVEN (( 1 + INDEX_BITS + LENGTH_BITS) / 9)
#define LOOK_AHEAD_SIZE ( RAW_LOOK_AHEAD_SIZE + BREAK_EVEN)
#define TREE_ROOT WINDOW_SIZE
#define END_OF_STREAM 0
#define UNUSED 0
#define MODULO(a) ( (a) & (WINDOW_SIZE - 1) )

char *CompressionName = "Кодек LZSS";
char *Usage = "входной_файл выходной_файл\n\n";

void fatal_error(char *str, ...)
{
    printf("Ошибка: %s\n", str);
    exit(1);
}

BFILE *OpenOutputBFILE(char *name)
{
    BFILE *bfile;

    bfile = (BFILE *)calloc(1, sizeof (BFILE));
    bfile->file = fopen(name, "wb");
    bfile->rack = 0;
    bfile->mask = 0x80;
    bfile->pacifier_counter = 0;
    return bfile;
}

BFILE *OpenInputBFile(char *name)
{
    BFILE *bfile;

    bfile = (BFILE *)calloc(1, sizeof (BFILE));
    bfile->file = fopen(name, "rb");
    bfile->rack = 0;
    bfile->mask = 0x80;
    bfile->pacifier_counter = 0;
    return bfile;
}

void CloseOutputBFile(BFILE *bfile)
{
    if (bfile->mask != 0x80)
        putc(bfile->rack, bfile->file);
    fclose(bfile->file);
    free((char *)bfile);
}

void CloseInputBFile(BFILE *bfile)

```

```

{
    fclose(bfile->file);
    free((char *)bfile);
}

void WriteBit(BFILE *bfile, int bit)
{
    if (bit)
        bfile->rack |= bfile->mask;
    bfile->mask >>= 1;
    if (bfile->mask == 0)
    {
        putc(bfile->rack, bfile->file);
        if ((bfile->pacifier_counter++ & PACIFIER_COUNT) == 0)
            putc('.', stdout);
        bfile->rack = 0;
        bfile->mask = 0x80;
    }
}

void WriteBits(BFILE *bfile, ulong code, int count)
{
    ulong mask;

    mask = 1L << (count - 1);
    while (mask != 0)
    {
        if (mask & code)
            bfile->rack |= bfile->mask;
        bfile->mask >>= 1;
        if (bfile->mask == 0)
        {
            putc(bfile->rack, bfile->file);
            if ((bfile->pacifier_counter++ & PACIFIER_COUNT)
== 0)
                putc('.', stdout);
            bfile->rack = 0;
            bfile->mask = 0x80;
        }
        mask >>= 1;
    }
}

int ReadBit(BFILE *bfile)
{
    int value;

    if (bfile->mask == 0x80)
    {
        bfile->rack = getc(bfile->file);
        if (bfile->rack == EOF)
            fatal_error("Ошибка в процедуре ReadBit!\n");
        if ((bfile->pacifier_counter++ & PACIFIER_COUNT) == 0)

```

```

        putc('.', stdout);
    }
    value = bfile->rack & bfile->mask;
    bfile->mask >>= 1;
    if (bfile->mask == 0)
        bfile->mask = 0x80;
    return (value ? 1 : 0);
}

ulong ReadBits(BFILE *bfile, int bit_count)
{
    ulong mask;
    ulong return_value;

    mask = 1L << (bit_count - 1);
    return_value = 0;
    while (mask != 0)
    {
        if (bfile->mask == 0x80)
        {
            bfile->rack = getc(bfile->file);
            if (bfile->rack == EOF)
                fatal_error("Ошибка в процедуре
ReadBits!\n");
            if ((bfile->pacifier_counter++ & PACIFIER_COUNT)
== 0)
                putc('.', stdout);
        }
        if (bfile->rack & bfile->mask)
            return_value |= mask;
        mask >>= 1;
        bfile->mask >>= 1;
        if (bfile->mask == 0)
            bfile->mask = 0x80;
    }
    return return_value;
}

void MainCompressFunc()
{
    char InputPath[1000];
    char OutputPath[1000];
    int add = 0;
    BFILE *output;
    FILE *input;
    printf("Введите адрес файла:\n");
    fflush(stdin);
    cin >> InputPath;
    printf("Введите адрес архива\n");
    fflush(stdin);
    cin >> OutputPath;

```

```

    input = fopen(InputPath, "rb");
    if (input == NULL)
        fatal_error("Ошибка при открытии файла для ввода\n");

    output = OpenOutputBFILE(OutputPath);
    if (output == NULL)
        fatal_error("Ошибка при открытии файла для вывода\n");

    printf("\nСжимаю %s в %s\n", InputPath, OutputPath);
    printf("Используя %s\n", CompressionName);
    CompressFile(input, output);
    CloseOutputBFile(output);
    fclose(input);
    print_ratios(InputPath, OutputPath);
}

void MainExpandFunc()
{
    char InputPath[1000];
    char OutputPath[1000];
    int add = 0;
    FILE *output;
    BFILE *input;
    printf("Введите адрес архива\n");
    fflush(stdin);
    cin >> InputPath;
    printf("Введите адрес выходного файла\n");
    cin >> OutputPath;
    input = OpenInputBFile(InputPath);

    if (input == NULL)
        fatal_error("Ошибка при открытии файла для ввода\n");
    output = fopen(OutputPath, "wb");
    if (output == NULL)
        fatal_error("Ошибка при открытии файла для вывода\n");
    printf("\nРаспаковываю %s в %s\n", InputPath, OutputPath);
    printf("Используя %s\n", CompressionName);

    ExpandFile(input, output);
    CloseOutputBFile(input);
    fclose(output);
}

void About()
{
    system("cls");
    printf("Это курсовой проект студента гр.830501\nГлилки  

    Андрея Владимировича\nАрхиватор файлов Windows кодировкой  

    LZSS\n");
    _getch();
    system("cls");
}

```

```

int main()
{

    char choice;

    setlocale(LC_ALL, "Russian");
    setbuf(stdout, NULL);
    setlocale(LC_ALL, "Russian");
    do
    {

        printf("\n1-Новый архив \n2-Распаковка \n3-О программе
\n4-Выход\n");
        fflush(stdin);
        choice = getchar();

        switch (choice)
        {
            case '1':
                MainCompressFunc();
                break;

            case '2':
                MainExpandFunc();
                break;

            case '3':
                About();
                break;

            case '4':
                exit(0);

            default:
                break;
        }
    } while (true);

    return 0;
}

void usage_exit(char *prog_name)
{
    char *short_name;
    char *extension;

    short_name = strrchr(prog_name, '\\');
    if (short_name == NULL)
        short_name = strrchr(prog_name, '/');
    if (short_name == NULL)
        short_name = strrchr(prog_name, ':');
    if (short_name != NULL)

```



```

        short_name++;
    else
        short_name = prog_name;
    extension = strchr(short_name, '.');
    if (extension != NULL)
        *extension = '\\0';
    printf("\\nПравильное использование: %s %s\\n",
        short_name, Usage);
    exit(0);
}

long file_size(char *name)
{
    long eof_ftell;
    FILE *file;

    file = fopen(name, "r");
    if (file == NULL)
        return (0L);
    fseek(file, 0L, SEEK_END);
    eof_ftell = ftell(file);
    fclose(file);
    return eof_ftell;
}

void print_ratios(char *input, char *output)
{
    long input_size;
    long output_size;
    int ratio;

    input_size = file_size(input);
    if (input_size == 0)
        input_size = 1;
    output_size = file_size(output);
    ratio = (int)(output_size * 100L / input_size);
    printf("\\nРазмер входного файла (байт): %ld\\n",
        input_size);
    printf("Размер выходного файла (байт): %ld\\n",
        output_size);
    if (output_size == 0)
        output_size = 1;
    printf("Степень сжатия: %d%%\\n", ratio);
}

uchar window[WINDOW_SIZE];

struct
{
    int parent;
    int smaller_child;
    int larger_child;
}

```

```

tree[WINDOW_SIZE + 1];

void InitTree(int r)
{
    tree[TREE_ROOT].larger_child = r;
    tree[r].parent = TREE_ROOT;
    tree[r].larger_child = UNUSED;
    tree[r].smaller_child = UNUSED;
}

void ContractNode(int old_node, int new_node)
{
    tree[new_node].parent = tree[old_node].parent;
    if (tree[tree[old_node].parent].larger_child == old_node)
        tree[tree[old_node].parent].larger_child = new_node;
    else
        tree[tree[old_node].parent].smaller_child = new_node;
    tree[old_node].parent = UNUSED;
}

void ReplaceNode(int old_node, int new_node)
{
    int parent;

    parent = tree[old_node].parent;
    if (tree[parent].smaller_child == old_node)
        tree[parent].smaller_child = new_node;
    else
        tree[parent].larger_child = new_node;
    tree[new_node] = tree[old_node];
    tree[tree[new_node].smaller_child].parent = new_node;
    tree[tree[new_node].larger_child].parent = new_node;
    tree[old_node].parent = UNUSED;
}

int FindNextNode(int node)
{
    int next;

    next = tree[node].smaller_child;
    while (tree[next].larger_child != UNUSED)
        next = tree[next].larger_child;
    return next;
}

void DeleteString(int p)
{
    int replacement;

    if (tree[p].parent == UNUSED)
        return;
    if (tree[p].larger_child == UNUSED)
        ContractNode(p, tree[p].smaller_child);
}

```

```

else
if (tree[p].smaller_child == UNUSED)
    ContractNode(p, tree[p].larger_child);
else
{
    replacement = FindNextNode(p);
    DeleteString(replacement);
    ReplaceNode(p, replacement);
}
}

int AddString(int new_node, int *match_pos)
{
    int i;
    int test_node;
    int delta;
    int match_len;
    int *child;

    if (new_node == END_OF_STREAM)
        return (0);

    test_node = tree[TREE_ROOT].larger_child;
    match_len = 0;

    for (;;)
    {
        for (i = 0; i < LOOK_AHEAD_SIZE; i++)
        {
            delta = window[MODULO(new_node + i)] -
                    window[MODULO(test_node + i)];
            if (delta != 0)
                break;
        }

        if (i >= match_len)
        {
            match_len = i;
            *match_pos = test_node;
            if (match_len >= LOOK_AHEAD_SIZE)
            {
                ReplaceNode(test_node, new_node);
                return match_len;
            }
        }

        if (delta >= 0)
            child = &tree[test_node].larger_child;
        else
            child = &tree[test_node].smaller_child;

        if (*child == UNUSED)
        {

```

```

        *child = new_node;
        tree[new_node].parent = test_node;
        tree[new_node].larger_child = UNUSED;
        tree[new_node].smaller_child = UNUSED;
        return match_len;
    }
    test_node = *child;
}

}

void CompressFile(FILE *input, BFILE *output)
{
    int i;
    int c;
    int look_ahead_bytes;
    int current_pos;
    int replace_count;
    int match_len;
    int match_pos;

    current_pos = 1;

    for (i = 0; i < LOOK_AHEAD_SIZE; i++)
    {
        if ((c = getc(input)) == EOF)
            break;
        window[current_pos + i] = (uchar)c;
    }

    look_ahead_bytes = i;
    InitTree(current_pos);
    match_len = 0;
    match_pos = 0;

    while (look_ahead_bytes > 0)
    {
        if (match_len > look_ahead_bytes)
            match_len = look_ahead_bytes;
        if (match_len <= BREAK_EVEN)
        {
            replace_count = 1;
            WriteBit(output, 1);
            WriteBits(output, (ulong>window[current_pos], 8);
        }
        else
        {
            WriteBit(output, 0);
            WriteBits(output, (ulong)match_pos, INDEX_BITS);
            WriteBits(output, (ulong)(match_len - (BREAK_EVEN
+ 1)),
                        LENGTH_BITS);
            replace_count = match_len;

```

```

    }

    for (i = 0; i < replace_count; i++)
    {
        DeleteString(MODULO(current_pos +
LOOK_AHEAD_SIZE));
        if ((c = getc(input)) == EOF)
            look_ahead_bytes--;
        else
            window[MODULO(current_pos + LOOK_AHEAD_SIZE)]
= (uchar)c;
        current_pos = MODULO(current_pos + 1);
        if (look_ahead_bytes)
            match_len = AddString(current_pos,
&match_pos);
    }

    WriteBit(output, 0);
    WriteBits(output, (ulong)END_OF_STREAM, INDEX_BITS);
}

void ExpandFile(BFILE *input, FILE *output)
{
    int i;
    int current_pos;
    int c;
    int match_len;
    int match_pos;

    current_pos = 1;

    for (;;)
    {
        if (ReadBit(input))
        {
            c = (int)ReadBits(input, 8);
            putc(c, output);
            window[current_pos] = (uchar)c;
            current_pos = MODULO(current_pos + 1);
        }
        else
        {
            match_pos = (int)ReadBits(input, INDEX_BITS);
            if (match_pos == END_OF_STREAM)
                break;
            match_len = (int)ReadBits(input, LENGTH_BITS);
            match_len += BREAK_EVEN;
            for (i = 0; i <= match_len; i++)
            {
                c = window[MODULO(match_pos + i)];
                putc(c, output);
                window[current_pos] = (uchar)c;
            }
        }
    }
}

```

```
        current_pos = MODULO(current_pos + 1);  
    }  
}  
}
```

**ПРИЛОЖЕНИЕ Г**  
*(обязательное)*  
Ведомость документов