

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Санкт-Петербургский государственный технологический институт  
(технический университет)»  
СПбГТИ(ТУ)

|  |  |  |
| --- | --- | --- |
| УГНС | 09.00.00 | Информатика и вычислительная техника |
| Направление подготовки | 09.03.03 | Прикладная информатика |
| Направленность (профиль) |  | Прикладная информатика в химии |
| Факультет |  | Информационных технологий и управления |
| Кафедра |  | Систем автоматизированного  проектирования и управления |
| Учебная дисциплина |  | **ОПЕРАЦИОННЫЕ СИСТЕМЫ** |

Курс 2 Группа 485

**Отчет по лабораторной работе № 2**

|  |  |
| --- | --- |
| **Тема:** | **Организация файловых систем** |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дмитриев Н.Д.

(подпись, дата) (инициалы, фамилия)

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Макарук Р.В.

(подпись, дата) (инициалы, фамилия)

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

Санкт-Петербург

2020

**1. Цель работы**

Знакомство с общими принципами организации файловых систем, проведение действий по подготовке к работе и эксплуатации жестких дисков, а также по работе с командами и утилитами операционной системы, предназначенными для этого.

**2. Аналитическая часть**

**2.1 Файловые системы**

NTFS — стандартная файловая система для семейства операционных систем Windows NT фирмы Microsoft. NTFS поддерживает хранение метаданных. Раздел NTFS, теоретически, может быть почти какого угодно размера. Максимальный размер раздела NTFS в данный момент ограничен лишь размерами жестких дисков. Как и любая другая система, NTFS делит все полезное место на кластеры - блоки данных, используемые единовременно. NTFS поддерживает почти любые размеры кластеров - от 512 байт до 64 Кбайт.

Каждый файл на томе NTFS представлен записью в специальном файле, называемом главной файловой таблицей (MFA — master file table). NTFS резервирует первые 16 записей таблицы для специальной информации. Первая запись этой таблицы описывает непосредственно главную файловую таблицу. За ней следует зеркальная запись (mirror record) MFT.

FAT32. Эта файловая система была разработана в 1996 году. Она также была ограничена в создании разделов, в частности, не более 32 Гб.

Пространство тома FAT32 логически разделено на три смежные области:

Зарезервированная область. Содержит служебные структуры, которые принадлежат загрузочной записи раздела (Partition Boot Record – PBR, для отличия от Master Boot Record – главной загрузочной записи диска; также PBR часто некорректно называется загрузочным сектором) и используются при инициализации тома;

Область таблицы FAT, содержащая массив индексных указателей ("ячеек"), соответствующих кластерам области данных. Обычно на диске представлено две копии таблицы FAT в целях надежности;

Область данных, где записано собственно содержимое файлов – т.е. текст текстовых файлов, кодированное изображение для файлов рисунков, оцифрованный звук для аудиофайлов и т.д. – а также метаданные – информация относительно имен файлов и папок, их атрибутов, времени создания и изменения, размеров и размещения на диске.

**Таблица 1 - Сравнение файловых систем NTFS и FAT**

|  |  |  |
| --- | --- | --- |
| Характеристика | NTFS | FAT16 и FAT32 |
| Размеры диска | 264 байт (16 экзабайт или 18 446 744 073 709 551 616 байт) | ~ 8 терабайт |
|  |  |  |
| Продолжение таблицы 1 | | |
| Размер тома | Теоретически — 264 минус 1 кластер. Разметка диска в стиле [MBR](http://ru.wikipedia.org/wiki/MBR) позволяет создавать разделы, размеры которых не превышают 2 ТБ. Чтобы обойти это ограничение, необходимо использование динамических дисков или разметки [GPT](http://ru.wikipedia.org/wiki/Таблица_разделов_GUID), поддерживающей разделы диска размером до 9.4 [ЗБ](http://ru.wikipedia.org/wiki/Зеттабайт) (9.4 × 1021 байт). | Запись в таблице FAT на томе с файловой системой FAT32 имеет размер 4 байта, поэтому средство [ScanDisk](http://ru.wikipedia.org/wiki/ScanDisk) не может работать с таблицей FAT на диске FAT32, описывающей более 4 177 920 кластеров (включая два резервных). С учетом самих таблиц FAT и при максимальном размере кластера 32 КБ размер тома может быть до 127,53 ГБ. |
| Форматирование дискет | Windows не позволяет форматировать дискеты в NTFS. |  |
| Поддержка типов ссылок | NTFS поддерживает жёсткие и [символьные](http://ru.wikipedia.org/wiki/Символьная_ссылка) ссылки, [Junctions](http://ru.wikipedia.org/wiki/NTFS_junction_point), [Volume Mount Point](http://ru.wikipedia.org/w/index.php?title=Volume_Mount_Point&action=edit&redlink=1). | FAT не поддерживает ссылки. |
| Максимальный размер файла | Теоретически — 264 байт минус 1 килобайт. Практически — 244 байт минус 64 килобайта (~16384 гигабайт или ~16 терабайт). | FAT16 поддерживает файлы размером не более 2 ГБ. FAT32 поддерживает файлы размером не более 4 ГБ. |
| Средства безопасности | Атрибуты файлов, [авторизация](http://ru.wikipedia.org/wiki/Авторизация) с использованием [DACL](http://ru.wikipedia.org/wiki/DACL), [шифрование](http://ru.wikipedia.org/wiki/Шифрование) с использованием [EFS](http://ru.wikipedia.org/wiki/EFS). | Атрибуты файлов. |
| Аудит | С использованием [SACL](http://ru.wikipedia.org/wiki/SACL). | Не поддерживается. |
| Поддержка сжатия | На уровне файловой системы для файлов, каталогов и дисков. | На уровне диска (в FAT16). В FAT32 не поддерживается. |
| Максимальное количество файлов | 4 294 967 295 (232 — 1). | В FAT32 не более 268 435 444 (228−12) |

Все ныне существующие причины повреждения дисков, как-то связанные с носителем и появившиеся на жёстком диске ошибки, или промежуточные повреждения, возникшие в памяти – всё это требует запуска служебной программы chkdsk.

Утилита, определяющая работоспособность NTFS прошлых версий, функционировала просто, согласно ей файловая система считалась или работоспособной или нет. К тому же проверяемый том во время проверки и исправления повреждений системы переводился в автономный режим.

**2.2 Дефрагментация**

Дефрагментация – это процесс перераспределения фрагментов файлов. Дефрагментацию делают для обеспечения непрерывной последовательности кластеров. Она однозначно положительно влияет на скорость работы системы.

Рассмотрим дефрагментацию диска на примере операционных систем Windows XP и Windows 10.

Сделать дефрагментацию диска или дисков под управлением 10 версии Windows достаточно просто. Нажмите на меню пуск в левом нижнем углу экрана и выбираем пункт «Компьютер». Откроется окно, в котором будут показаны все ваши диски.

Выберите диск, дефрагментацию которого следует выполнить. Наведите мышкой на требуемый диск и щёлкните правой кнопкой мыши.

Откроется контекстное меню, в котором надо выбрать пункт «Свойства».

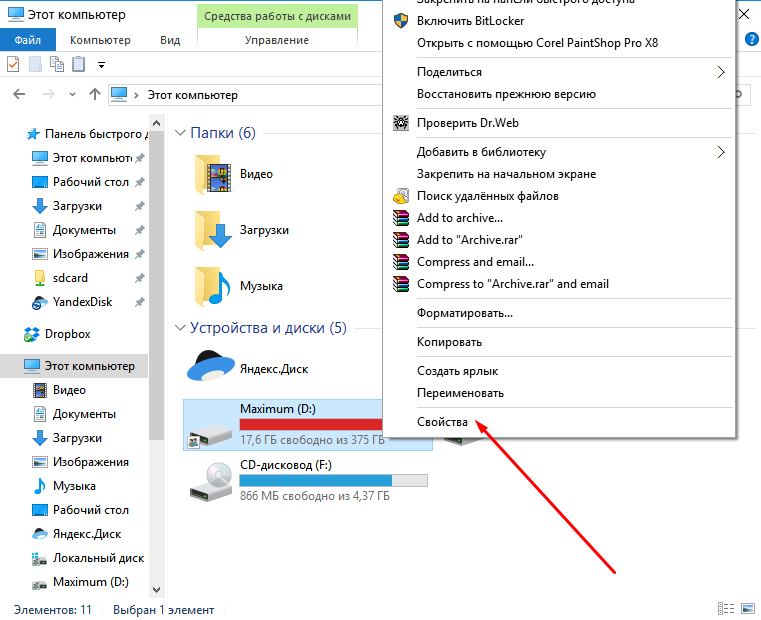


Рисунок 1 – Контекстное меню диска

Откроется окно, в котором будет 7 вкладок. Выберите «Сервис», пункт «Оптимизировать».

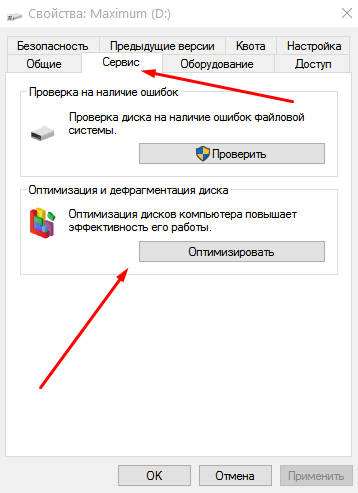


Рисунок 2 – Свойства локального диска

Откроется новое окно с дефрагментатором, который встроена в систему Windows 10.

Здесь есть несколько пунктов:

* «Анализировать» - анализирует выбранный диск на уровень дефрагментации.
* «Оптимизировать» - запускает процесс дефрагментации выбранного диска.
* «Изменить параметры» - параметры запуска автоматической дефрагментации дисков. Например, ежедневно, раз в неделю и т.п.

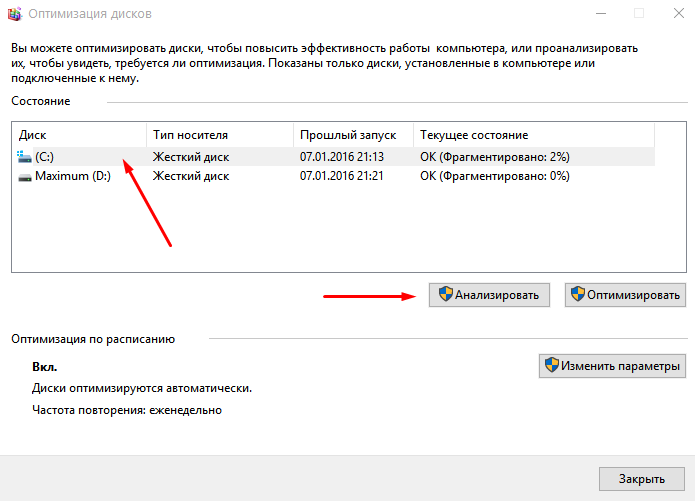


Рисунок 3 – Внешний вид программы дефрагментации Windows 10

Если вы давно не проводили дефрагментацию, то вкладка «Анализировать диск» не нужна.

Во время дефрагментации диска не следует пользоваться компьютером. Это чревато зависаниями, особенно если у вас слабый ПК или ноутбук. Длительность зависит от объёма и заполненности жесткого диска компьютера.

Теперь рассмотрим дефрагментацию диска на Windows XP.

Принцип запуска программы дефрагментации схож с Windows 10:

1. «Мой Компьютер», выбираем нужный диск, нажимаем правую кнопку, и выбираем со списка «Свойства».
2. Выбираете «Сервис» и жмем «Выполнить дефрагментацию…».
3. Выбираете нужный диск, на котором нужно сделать дефрагментацию. Нажимаете «Анализ», и жмем «Дефрагментация».

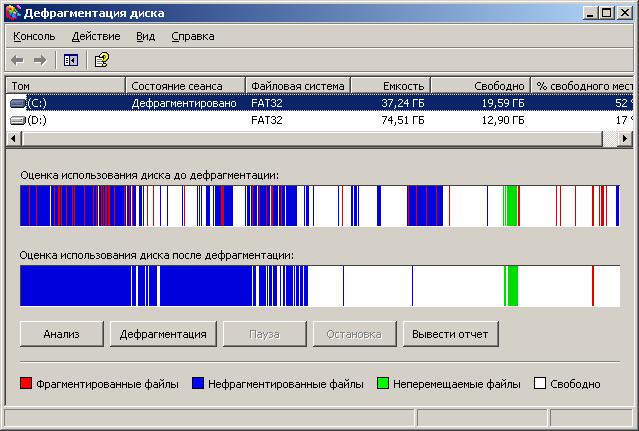


Рисунок 4 – Внешний вид программы дефрагментации на Windows XP

**Виды ошибок**

1. Пересечение кластеров - ситуация, в которой два файла замыкаются на одном и том же кластере.
2. "BAD" кластеры – кластеры, на которых находятся физические или логические ошибки, мешающие записи данных в данный кластер.
3. Незаконченные файлы - в кластере отсутствует ссылка на конец файла.
4. Потерянные файлы – файлы, которые занимают определенные кластеры, на которые нет ссылки.

**3. Ход работы**

Код программы:

[файл Program.cs]

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lab2

{

class Program

{

static readonly string NL = Environment.NewLine;

static void OutputInConsole(List<File> files, List<Cluster> badClusters,

Dictionary<Cluster, List<File>> crossingClustersAndFiles, List<Cluster> lostClusters)

{

foreach (var file in files)

Console.WriteLine(file.GetFileInStr() + NL);

if (badClusters.Any())

{

Console.WriteLine("Bad-кластеры:");

foreach (var badCluster in badClusters)

Console.WriteLine(badCluster);

}

if (crossingClustersAndFiles.Any())

{

Console.WriteLine(NL + "Пересекающиеся кластеры:");

foreach (var crossingCluster in crossingClustersAndFiles.Keys)

{

Console.WriteLine(crossingCluster + " Файлы, задействующие этот кластер:");

foreach (var file in crossingClustersAndFiles[crossingCluster])

Console.WriteLine(file.FileName);

}

}

if (lostClusters.Any())

{

Console.WriteLine(NL + "Потерянные кластеры:");

foreach (var missingCluster in lostClusters)

Console.WriteLine(missingCluster);

}

}

static void Main()

{

var filesAndClusters = FileWorker.GetFilesAndClusters();

var files = filesAndClusters.Item1;

var clusters = filesAndClusters.Item2;

files = FileWorker.LinkClustersToFiles(files, clusters);

var badClusters = ClusterChecker.GetBadClusters(clusters);

var crossingClustersAndFiles = ClusterChecker.GetCrossingClustersAndFiles(files);

var lostClusters = ClusterChecker.GetMissingClusters(files, clusters);

OutputInConsole(files, badClusters, crossingClustersAndFiles, lostClusters);

}

}

}

[конец файла Program.cs]

[файл Cluster.cs]

using static Lab2.ClusterType;

namespace Lab2

{

public class Cluster

{

public int ClusterNum { get; }

public int NextClusterNum { get; }

public string ClusterType { get; set; }

public Cluster(string clusterStr, int clusterNum)

{

SetClusterType(clusterStr);

if (this.ClusterType == Normal)

this.NextClusterNum = int.Parse(clusterStr);

ClusterNum = clusterNum;

}

public static implicit operator string(Cluster cluster) =>

$"Кластер {cluster.ClusterNum}. Тип \"{cluster.ClusterType}\".";

public static bool operator !=(Cluster cluster1, Cluster cluster2)

{

return !(cluster1 == cluster2);

}

public static bool operator ==(Cluster cluster1, Cluster cluster2)

{

return cluster1.ClusterNum == cluster2.ClusterNum;

}

private void SetClusterType(string clusterStr)

{

string loweredClusterStr = clusterStr.ToLower();

if (loweredClusterStr.Contains("bad"))

ClusterType = Bad;

else if (loweredClusterStr.Contains("eof"))

ClusterType = Eof;

else if (int.TryParse(loweredClusterStr, out int \_))

ClusterType = Normal;

else

ClusterType = Empty;

}

}

public static class ClusterType

{

public static string Normal => "Нормальный кластер";

public static string Bad => "Bad-кластер";

public static string Eof => "Конец файла";

public static string Empty => "Пустой кластер";

}

}

[конец файла Cluster.cs]

[файл ClusterChecker.cs]

using System.Collections.Generic;

using System.Linq;

using static Lab2.ClusterType;

namespace Lab2

{

public static class ClusterChecker

{

public static List<Cluster> GetBadClusters(List<Cluster> clusters)

{

var badClusters = new List<Cluster>();

foreach (var cluster in clusters)

if (cluster.ClusterType == Bad)

badClusters.Add(cluster);

return badClusters;

}

public static Dictionary<Cluster, List<File>> GetCrossingClustersAndFiles(List<File> files)

{

var crossingFilesAndClusters = new Dictionary<Cluster, List<File>>();

foreach (var file in files)

{

foreach (var cluster in file.Clusters)

{

// проверка, содержит ли уже словарь выбранный кластер

if (!crossingFilesAndClusters.Any(d => d.Key == cluster))

{

var crossingFiles = files.Where(f => f.Clusters.Any(c => c == cluster) && f != file)

.ToList();

if (crossingFiles.Any())

crossingFilesAndClusters.Add(cluster, crossingFiles.Prepend(file).ToList());

}

}

}

return crossingFilesAndClusters;

}

public static List<Cluster> GetMissingClusters(List<File> files, List<Cluster> clusters)

{

var missingClusters = new List<Cluster>();

foreach (var cluster in clusters)

{

if (!files.Any(f => f.Clusters.Any(c => c == cluster)) && cluster.ClusterType != Empty)

missingClusters.Add(cluster);

}

return missingClusters;

}

}

}

[конец файла ClusterChecker.cs]

[файл File.cs]

using System;

using System.Collections.Generic;

using static Lab2.ClusterType;

namespace Lab2

{

public class File

{

readonly string NL = Environment.NewLine;

public int FirstClusterNum { get; }

public List<Cluster> Clusters { get; } = new List<Cluster>();

public string FileName { get; }

public File(string fileName, string firstClusterNum)

{

FileName = fileName;

FirstClusterNum = int.Parse(firstClusterNum);

}

public string GetFileInStr()

{

string fileStr = $"Файл \"{FileName}\":" + NL;

foreach (var cluster in Clusters)

{

fileStr += cluster;

fileStr += NL;

if (cluster.ClusterType == Eof)

{

fileStr += $"Файл \"{FileName}\" закончен на {cluster.ClusterNum} кластере и не повреждён.";

}

else if (cluster.ClusterType != Normal)

{

fileStr += $"Файл \"{FileName}\" повреждён, так как тип его последнего кластера " +

$"\"{cluster.ClusterType}\".";

break;

}

}

return fileStr;

}

}

}

[конец файла File.cs]

[файл FileWorker.cs]

using System;

using System.Collections.Generic;

using System.IO;

using static Lab2.ClusterType;

namespace Lab2

{

public static class FileWorker

{

public static Tuple<List<File>, List<Cluster>> GetFilesAndClusters()

{

var files = new List<File>();

var clusters = new List<Cluster>();

using (var reader = new StreamReader("1.txt"))

{

bool filesTableRead = false;

while (!filesTableRead)

{

var str = reader.ReadLine();

if (str.Contains("."))

{

var file = str.Split('.');

var fileName = file[0];

var firstClusterNum = file[1];

files.Add(new File(fileName, firstClusterNum));

}

else

filesTableRead = true;

}

int clustersCounter = 0;

while (!reader.EndOfStream)

{

var cluster = reader.ReadLine();

clusters.Add(new Cluster(cluster, clustersCounter));

++clustersCounter;

}

}

return new Tuple<List<File>, List<Cluster>>(files, clusters);

}

private static void LinkClustersToFile(File file, Cluster cluster, List<Cluster> clusters)

{

file.Clusters.Add(cluster);

if (cluster.ClusterType == Normal)

{

var nextClusterNum = cluster.NextClusterNum;

var nextCluster = clusters[nextClusterNum];

LinkClustersToFile(file, nextCluster, clusters);

}

}

public static List<File> LinkClustersToFiles(List<File> files, List<Cluster> clusters)

{

foreach (var file in files)

{

var firstCluster = clusters[file.FirstClusterNum];

LinkClustersToFile(file, firstCluster, clusters);

}

return files;

}

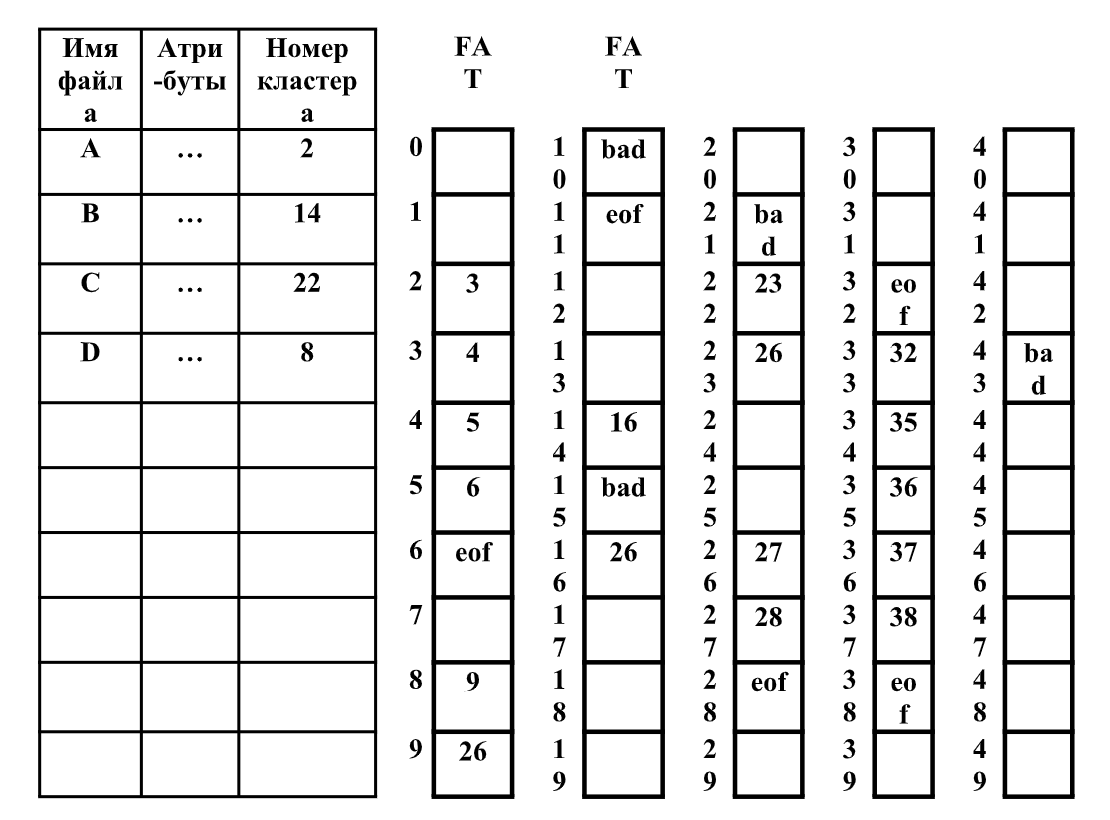
}

}

[конец файла FileWorker.cs]

**4. Работа программы**

На вход программе были предоставлены следующие данные (Рисунок 5).

Рисунок 5 – Данные из задания

Результаты работы программы при введённых выше данных продемонстрированы на Рисунках 6, 7, 8.

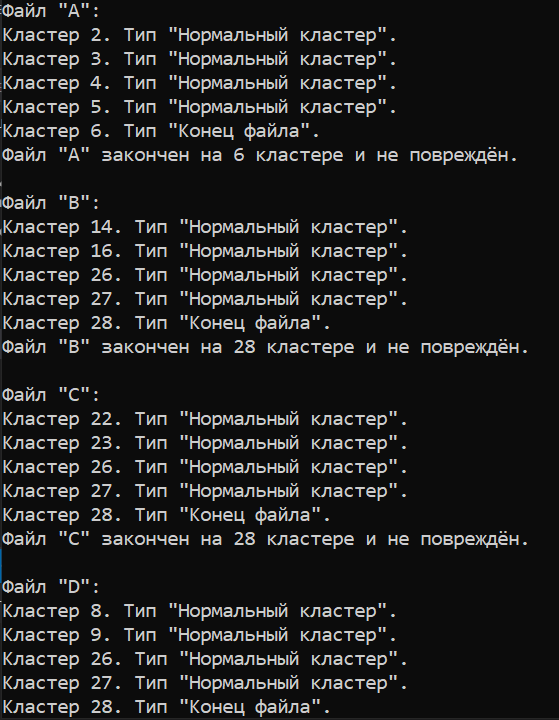


Рисунок 6 – Результаты работы программы

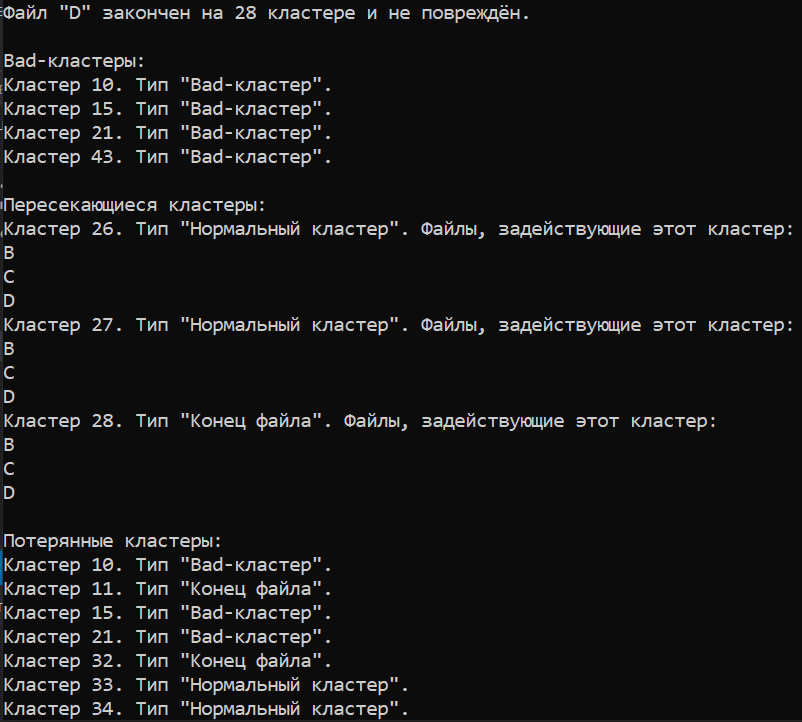


Рисунок 7 – Результаты работы программы

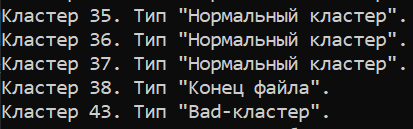


Рисунок 8 – Результаты работы программы

**5. Ответы на контрольные вопросы**

**В чем отличие физической и логической конфигурации жесткого диска?**

Логическая конфигурация описывает способ представления данных во внешней памяти с точки зрения пользователя и приложений. Элементами логической структуры являются файлы и каталоги.

Физическая структура накопителя определяет элементы хранения данных - физические блоки, предоставляемые для размещения данных файлов и каталогов. Размеры и особенности использования физических блоков зависят от типа накопителя. Ключевым моментом в организации файловой системы является способ размещения элементов логической структуры файловой системы (файлов, каталогов) на конкретном накопителе с известными физическими характеристиками. Такую задачу называют отображением логической структуры файловой системы на физическую структуру накопителя.

**В чем суть стратегии несвязного размещения данных на диске? В чем ее преимущество перед связным способом?**

При несвязном распределении внешней памяти – используется распределение с помощью списков секторов. Несвязное распределение - более динамично. В этом случае дисковая память рассматривается, как набор индивидуальных секторов, которые могут находиться в различных местах дисковой памяти. Секторы, принадлежащие одному файлу, содержат ссылки друг на друга, образуя список. В списке свободного пространства содержаться все свободные секторы дисковой памяти. Если необходимо увеличить размер файла - запрашиваются свободные сектора, если файл уменьшается, то он возвращает освободившиеся сектора в списки свободных. Достоинства: не возникает проблем уплотнения. Недостатки: поиск может быть сопряжен с длительными поисками подводами (особенно для логически смежных записей).

**6. Вывод**

Во время выполнения лабораторной работы были проведены действия по работе c программой дефрагментации жестких дисков, на примере Windows XP и Windows 10, а также по работе с командами и утилитами операционной системы, предназначенными для этого.

Полученные сведения помогли расширить возможности прикладного программирования.

Ошибками файловой системы являются:

* Пересечение кластеров
* Потерянные кластеры
* Незаконченные файлы

Данный алгоритм имитирует работу сканировщика диска, указывая ошибки файловой системы.