**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский государственный университет экономики, статистики и информатики (МЭСИ)»**

Курсовая работа

Дисциплина «Объектно-ориентированное программирование»

на тему:

**«Анализ и проверка файловых систем»**

Выполнил:

Студент 2-го курса ДКО 202

Вишняков Ф. Н.

Проверил:

Черепанов А.А.

Москва 2013

Contents

[Введение 4](#_Toc376073800)

[Постановка задачи 7](#_Toc376073801)

[Описание программы 8](#_Toc376073802)

[Код программы 11](#_Toc376073803)

[Catalogue.h 11](#_Toc376073804)

[FileSystem.h 18](#_Toc376073805)

[Приложение А: длинная арифметика 28](#_Toc376073806)

[digit.h 28](#_Toc376073807)

[digit10.h 28](#_Toc376073808)

[Long.h 30](#_Toc376073809)

[Приложение B: Тестирующий код программы main.cpp 38](#_Toc376073810)

[Литература 40](#_Toc376073811)

# Введение

Файловая система (англ. file system) — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов и (каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Практически всегда файлы на дисках объединяются в каталоги.

В простейшем случае все файлы на данном диске хранятся в одном каталоге. Такая одноуровневая схема использовалась в CP/M и в первой версии MS-DOS 1.0. Иерархическая файловая система со вложенными друг в друга каталогами впервые появилась в Multics, затем в UNIX.

Каталоги на разных дисках могут образовывать несколько отдельных деревьев, как в DOS/Windows, или же объединяться в одно дерево, общее для всех дисков, как в UNIX-подобных системах.В UNIX существует только один корневой каталог, а все остальные файлы и каталоги вложены в него. Чтобы получить доступ к файлам и каталогам на каком-нибудь диске, необходимо смонтировать этот диск командой mount. Например, чтобы открыть файлы на CD, нужно, говоря простым языком, сказать операционной системе: «возьми файловую систему на этом компакт-диске и покажи её в каталоге /mnt/cdrom». Все файлы и каталоги, находящиеся на CD, появятся в этом каталоге /mnt/cdrom, который называется точкой монтирования. В большинстве UNIX-подобных систем съёмные диски (дискеты и CD), флеш-накопители и другие внешние устройства хранения данных монтируют в каталог /mnt, /mount или /media. Unix и UNIX-подобные операционные системы также позволяют автоматически монтировать диски при загрузке операционной системы.

По предназначению файловые системы можно классифицировать на нижеследующие категории.

* Для носителей с произвольным доступом (например, жёсткий диск): FAT32, HPFS, ext2 и др. Поскольку доступ к дискам в разы медленнее, чем доступ к оперативной памяти, для прироста производительности во многих файловых системах применяется асинхронная запись изменений на диск. Для этого применяется либо журналирование, например в ext3, ReiserFS, JFS, NTFS, XFS, либо механизм soft updates и др. Журналирование широко распространено в Linux, применяется в NTFS. Soft updates — в BSD системах.
* Для носителей с последовательным доступом (например, магнитные ленты): QIC и др.
* Для оптических носителей — CD и DVD: ISO9660, HFS, UDF и др.
* Виртуальные файловые системы: AEFS и др.
* Сетевые файловые системы: NFS, CIFS, SSHFS, GmailFS и др.
* Для флэш-памяти: YAFFS, ExtremeFFS, exFAT.

Немного выпадают из общей классификации специализированные файловые системы: ZFS (собственно файловой системой является только часть ZFS), VMFS (т. н. кластерная файловая система, которая предназначена для хранения других файловых систем) и др.

Ограничения, использующиеся в работе:

Максимальный размер файла:

* NTFS - 2^64 байт
* FAT32 - 4 ГБ
* EXT3 - 2^40 байт
* EXT4 - 2^44 байт
* DVD - 4.7 ГБ

Имя файла

В ФС Windows:

* запрещены символы NULL / \ | " ? \* > < :
* запрещены имена con, aux, prn, nul, com[1-9], lpt[1-9]
* запрещены пробелы в начале/конце названия файла

В UNIX:

* запрещены символы NULL, /

В DVD:

* название содержит лишь латинские буквы верхнего регистра, цифры, точку
* в начале и конца названия файла не может содержаться точка
* в названии файла должна быть максимум одна точка
* в названии каталога не должно содержаться точек

Длина названия файла/директории:

* Windows: 255
* UNIX: 256
* DVD: 8.3 для файлов и 8 для директории

Вложенность:

DVD - максимальная глубина каталога = 8

# Постановка задачи

Реализовать классы для хранения файлов и директорий в каталоге. Создать при помощи классов средства для проверки и анализа файлов и директорий, а именно возможность расположения каталога в определенной файловой системе (использующуюся в Windows/UNIX), возможность записи каталога на DVD. Проверять ограничения, присущие определенным файловым системам (размер файла, длина названия, запрещенные имена и т.д.), искать файлы с атрибутами

Дополнительно, с помощью добавления к моему проекту проекта одногруппника по реализации длинной арифметики и небольшими дополнениями к ней (реализация интерфейса, более расположенного к пользователю), стало возможным сравнивать точным образом размеры файлов, не ограничиваясь встроенными в C++ лимитами чисел.

# Описание программы

Курсовая работа делится на два связанных файла: Catalogue.h и FileSystem.h. В каталоге описаны структуры, которые определяют параметры файлов:

struct Attributes

{

bool invisible;

bool readOnly;

}

Эта структура содержит флаги скрытности и атрибута "только для чтения".

struct Param

{

Long size;

int depth;

string name;

Attributes attributes;

}

Здесь описаны стандартные данные, такие как размер файла и его имя. Также здесь расположен служебный параметр, который вычисляется после создания иерархии ФС: глубина вложенности. Он используется для вывода и для определения некоторых ограничений.

Обе структуры содержат конструкторы для упрощения создания элементов.

Далее идет класс Elem. Это класс предназначен для содержания элементов списка List, реализованного ниже. Он практически пустой: содержит лишь конструкторы, необходимые для создания файлов и папок, служебные методы "геттеры" и один абсолютно виртуальный метод для вывода.

От этого класса наследуются классы File и Directory - основа программы. Класс для файла содержит лишь конструкторы и перегрузку абсолютно виртуального метода вывода из класса Elem. Файл для директорий же, помимо этого, содержит ещё и список, содержащий другие файлы и папки, дополнительный метод для вывода, требующийся для отображения содержимого папки. Также методы добавления элементов в список этой директории: по одному или по фиксированному указанному количеству, что иногда бывает удобно. Ещё класс содержит метод для вычисления "глубин" файлов и каталогов, а также небольшой интерфейс для этого метода.

Класс Iter создан для удобной навигации внутри списка. В нем реализован конструктор по умолчанию и методы и перегруженные операторы для написания подобных вещей:

for (Iter i = begin(); i != end(); ++i)

Класс List является "внутренностью" класса Directory и содержит в себе методы задания итераторов, вставки/удаления элементов из списка, определения количества элементов внутри списка, а также служебный метод, возвращающий ссылку на первый элемент.

Заголовочный файл FileSystem.h, в свою очередь, содержит обширный набор классов, совершающих проверку и анализ файловой системы.

Класс FileCheck является базовым классом для двух других классов проверок ФС семейств Windows и UNIX. Задание файла для проверки происходит при помощи указания в конструкторе ссылки на класс File или приравнивание к этому указателю класса. Таким образом, для каждого сета проверок достаточно иметь один экземпляр тестирующего класса.

В классе реализованы простые проверки на скрытость и RO файла.

Класс DirectoryCheck является базовым для AttributeCheck и DVDDirectoryCheck. Первый ищет количество скрытых/RO файлов, второй определяет, можно ли записать каталог и все вложенные в него файлы/каталоги на DVD-диск.

Об ограничениях, проверяемых в этих классах, подробнее написано во вступлении.

К заимствованным классам длинной арифметики были добавлены следующие методы и функции:

Long involution(long);

Вовзведение длинного числа в степень. Степень указывается в параметрах метода, сам метод возвращает результат и записывает его в текущем объекте.

bool empty();

Возвращает истину, если число в объекте равно 0.

Long assign(Long &l, string b)

Дружественная функция, переводит число в строке string в длинное число Long. Возвращает объект l.

Long& operator\*(const Long& a, string b)

Дружественная функция для умножения длинного числа на число из строки.

bool operator>(const Long& a, string b)

Перегруженный оператор сравнения для длинного числа и строки.

# Код программы

## Catalogue.h

#pragma once

#include <iostream>

#include <string>

#include <stdarg.h>

#include "Long.h"

using namespace std;

struct Attributes

{

bool invisible;

bool readOnly;

Attributes operator=(Attributes a)

{

invisible = a.invisible;

readOnly = a.readOnly;

return \*this;

}

};

struct Param

{

Long size;

int depth;

string name;

Attributes attributes;

Param() {}

Param(Long s, string n, bool i, bool r)

{

size = s;

name.assign(n);

attributes.invisible = i;

attributes.readOnly = r;

depth = 0;

}

Param operator=(Param p)

{

attributes = p.attributes;

size = p.size;

name.assign(p.name);

depth = 0;

return p;

}

};

class Elem

{

friend class List;

friend class Directory;

protected:

Param param;

Elem\* next;

public:

Param getParam()

{

return param;

}

Long getSize()

{

return param.size;

}

string getStr()

{

return param.name;

}

Elem(Param p)

{

param = p;

next = NULL;

}

Elem(Long size, string name, bool invisible, bool readOnly)

{

param = Param(size, name, invisible, readOnly);

next = NULL;

}

Elem\* next\_elem()

{

return next;

}

virtual string str() = 0;

};

class Iter

{

friend class List;

Elem\* current\_elem;

public:

Iter() : current\_elem(0) {}

Iter operator++()

{

if (current\_elem)

current\_elem = current\_elem -> next\_elem();

return \*this;

}

Elem& operator\*()

{

return \*current\_elem;

}

Elem\* operator->()

{

return current\_elem;

}

bool operator!=(Iter v)

{

return current\_elem != v.current\_elem;

}

Elem\* operator+()

{

return current\_elem -> next\_elem();

}

};

class List

{

protected:

friend class Iter;

Elem\* first;

public:

List() : first(NULL) {};

~List() { }

Elem\* getFirst()

{

return first;

}

int length()

{

int ret = 0;

Elem\* temp = first;

while (temp)

{

ret++;

temp = temp -> next;

}

return ret;

}

void push\_back(Elem\* p)

{

Elem \*e = NULL;

for (Iter i = begin(); i != end(); ++i)

e = &\*i;

if (e)

e -> next = p;

else

first = p;

p -> next = NULL;

}

Elem\* pop()

{

Elem \*p = first;

if (p)

first = p->next;

return p;

}

Iter begin()

{

Iter tmp;

tmp.current\_elem = first;

return tmp;

}

Iter end()

{

Iter tmp;

tmp.current\_elem = NULL;

return tmp;

}

};

class File : public Elem

{

public:

File(Param p) : Elem(p) {}

File(Long size, string name, bool invisible, bool readOnly) :

Elem(size, name, invisible, readOnly) {}

File(File \*f) : Elem(f->param) {}

string str()

{

string s;

for (int i = 0; i < param.depth; i++)

s += " ";

s += param.name;

return s;

}

};

class Directory : public Elem

{

List content;

friend class DVDDirectoryCheck;

public:

Directory(Param p) : Elem(p) {}

Directory(Long size, string name, bool invisible, bool readOnly) :

Elem(size, name, invisible, readOnly) {}

Directory(Directory \*f) : Elem(f->param) {}

List getContent()

{

return content;

}

string content\_str()

{

string s;

for (Iter i = content.begin(); i != content.end(); ++i)

s += i -> str() + ((+i)!=NULL?'\n':' ');

return s;

}

string str()

{

string s;

for (int i = 0; i < param.depth; i++)

s += " ";

return s + param.name + "/\n" + content\_str();

}

Directory& push\_back(int n, ...)

{

va\_list elements;

va\_start(elements, n);

for (int i = 0; i < n; i++)

content.push\_back(va\_arg(elements, Elem\*));

va\_end(elements);

return \*this;

}

Directory& push\_back(Elem\* a)

{

content.push\_back(a);

return \*this;

}

void deepness(Directory\* a, File\* el)

{

if (el == NULL)

if (a->getContent().getFirst())

{

a->getContent().getFirst()->param.depth=a->param.depth+1;

if ((a->getContent().getFirst())->getSize().empty())

deepness(dynamic\_cast<Directory\*>(a->getContent().getFirst()), NULL);

else

deepness(NULL, dynamic\_cast<File\*>(a->getContent().getFirst()));

}

if (el && el->next\_elem())

{

el->next\_elem()->param.depth = el->param.depth;

if (!((el->next\_elem()->getSize().empty())))

deepness(NULL, dynamic\_cast<File\*>(el->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(el->next\_elem()), NULL);

}

if (a && a -> next\_elem())

{

a->next\_elem()->param.depth = a -> param.depth;

if (!((a->next\_elem()->getSize().empty())))

deepness(NULL, dynamic\_cast<File\*>(a->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(a->next\_elem()), NULL);

}

}

void prepare()

{

deepness(this, NULL);

}

};

ostream &operator<<(ostream &out, Elem &e)

{

out << e.str();

return out;

}

## FileSystem.h

#pragma once

#include <algorithm>

#include "Catalogue.h"

class FileCheck

{

protected:

File\* toCheck;

public:

void operator=(File\* f)

{

toCheck = (new File(f));

}

FileCheck(File\* toCheck) : toCheck(toCheck) {};

virtual bool summarize() = 0;

bool invisible()

{

Param a = toCheck -> getParam();

if (a.attributes.invisible)

{

cout << "File is invisible\n";

return true;

}

return false;

}

bool readOnly()

{

Param a = toCheck -> getParam();

if (a.attributes.readOnly)

{

cout << "File is only for reading";

return true;

}

return false;

}

};

class WindowsFileCheck : public FileCheck

{

static string restrictedNames[4];

static char restrictedSymbols[10];

public:

WindowsFileCheck(File \*toCheck) : FileCheck(toCheck) {}

bool summarize()

{

cout << "\nChecking file " << toCheck -> getStr() << "...\n";

if (!(restrictedNameCheck() || fileNameCheck() || fileSizeCheck()))

{

cout << toCheck -> getStr() << " has no errors\n";

return true;

}

return false;

}

bool restrictedNameCheck()

{

string s = toCheck -> getStr().substr(0, 3);

string ss = toCheck -> getStr();

transform(s.begin(), s.end(), s.begin(), ::tolower);

bool bad = false;

for (int i = 0; i < 4; i++)

if (s == restrictedNames[i])

bad = true;

if (ss.length() > 3 && ((int)ss[3]) <= 58 &&

((int)ss[3]) >= 48 && (s == "com" || s == "lpt"))

bad = true;

if (bad)

cout << "Restricted in NTFS/FAT32 name\n";

return bad;

}

bool fileNameCheck()

{

string badS("");

string ss = toCheck -> getStr();

bool bad = false;

for (int i = 1; i < 10; i++)

if (ss.find(restrictedSymbols[i]) != ss.npos)

{

bad = true;

badS += restrictedSymbols[i];

}

for (int i = 0; i < ss.length(); i++)

if (ss[i] == '\0')

badS += "NULL ";

if (bad)

cout << "Filename contains bad NTFS/FAT32 symbols: " << badS << "\n";

if (ss.length() > 255)

{

bad = true;

cout << "Filename contains more than 255 symbols\n";

}

if (ss[0] == ' ' || ss[ss.length() - 1] == ' ')

{

cout << "Filename contains spaces at the beginning/end\n";

bad = true;

}

return bad;

}

bool fileSizeCheck()

{

bool bad = false;

Long NTFS, FAT32, size;

assign(NTFS, "2");

NTFS.involution(64);

assign(FAT32, "2");

FAT32.involution(10);

NTFS = NTFS - FAT32;

assign(FAT32, "2");

FAT32.involution(32);

size = toCheck -> getSize();

if (size > FAT32)

{

bad = true;

cout << "FAT32 cannot contain this file due to its size\n";

}

if (size > NTFS)

{

bad = true;

cout << "NTFS cannot contain this file due to its size\n";

}

return bad;

}

};

class UnixFileCheck : public FileCheck

{

public:

UnixFileCheck(File \*toCheck) : FileCheck(toCheck) {}

bool summarize()

{

cout << '\n' << toCheck -> getStr() << '\n';

if (!(fileNameCheck() || fileSizeCheck()))

{

cout << "Has no errors\n";

return true;

}

return false;

}

bool fileNameCheck()

{

string badS("");

string ss = toCheck -> getStr();

bool bad = false;

if (ss.find('/') != ss.npos)

{

bad = true;

badS += '/';

}

for (int i = 0; i < ss.length(); i++)

if (ss[i] == '\0')

badS += "NULL ";

if (bad)

cout << "Filename contains bad EXT3/EXT4 symbols: " << badS << "\n";

if (ss.length() > 256)

{

bad = true;

cout << "Filename contains more than 256 symbols\n";

}

return bad;

}

bool fileSizeCheck()

{

bool bad = false;

Long EXT3, EXT4, size;

assign(EXT3, "2");

EXT3.involution(40);

assign(EXT4, "2");

EXT4.involution(44);

size = toCheck -> getSize();

if (size > EXT3)

{

bad = true;

cout << "EXT3 cannot contain this file due to its size\n";

}

if (size > EXT4)

{

bad = true;

cout << "EXT4 cannot contain this file due to its size\n";

}

return bad;

}

};

class DVDFileCheck : public FileCheck

{

public:

DVDFileCheck(File \*toCheck) : FileCheck(toCheck) {}

bool summarize(bool dir)

{

cout << "\nChecking " << (dir?"directory ":"file ") << toCheck -> getStr() << "...\n";

if ((!size() && !symbols() && !dotStartEnd() && !dotCount(dir)) &&

!length(dir))

{

cout << toCheck -> getStr() << " can be written on DVD\n";

return true;

}

else

{

cout << toCheck -> getStr() << " cannot be written on DVD\n";

return false;

}

}

bool summarize()

{

return summarize(false);

}

bool size()

{

Long size;

Long DVD;

assign(DVD, "10");

DVD.involution(8);

DVD = DVD \* assign(size, "47");

size = toCheck->getSize();

if (size> DVD)

cout << "My file is too big\n";

return size > DVD;

}

bool length(bool dir)

{

bool bad = dir?toCheck ->getStr().length() > 8:toCheck ->getStr().length() > 11;

if (bad)

cout << "Filename is too long\n";

return bad;

}

bool symbols()

{

char buf[1024];

strcpy(buf, toCheck->getStr().c\_str());

bool bad = false;

for (int i = 0; i < strlen(buf); i++)

if (!((buf[i] >= 'A' && buf[i] <= 'Z') || (buf[i] >= '0' && buf[i] <= '9') ||

buf[i] == '.' || buf[i] == ' '))

bad = true;

if (bad)

cout << "Filename contains bad ISO 9660 symbols\n";

return bad;

}

bool dotCount(bool dir)

{

string ss = toCheck -> getStr();

int c = 0;

for (int i = 0; i < ss.length(); i++)

if (ss[i] == '.')

c++;

if (dir)

return c > 0;

else

return c > 1;

}

bool dotStartEnd()

{

bool a = toCheck -> getStr()[0] == '.' ||

toCheck -> getStr()[toCheck -> getStr().length() - 1] == '.';

if (a)

cout << "Filename has dot at the end/beginning\n";

return a;

}

};

class DirectoryCheck

{

protected:

Directory\* toCheck;

public:

DirectoryCheck(Directory\* toCheck) : toCheck(toCheck) {};

void operator=(Directory\* d)

{

toCheck = (new Directory(d));

}

virtual void summarize() = 0;

};

class AttributeCheck : public DirectoryCheck

{

public:

AttributeCheck(Directory \*toCheck) : DirectoryCheck(toCheck) {}

void summarize()

{

int i = 0, r = 0, ir = 0;

cout << "Searching for invisible/RO files in " << toCheck->getParam().name << "...\n";

deepness(toCheck, i, r, ir, NULL);

cout << "Invisible files: " << i << "\nRO files: " << r << "\nInvisible/RO files: "

<< ir << endl;

}

void deepness(Directory\* a, int &i, int &r, int &ir, File\* el)

{

if (el == NULL)

if (a->getContent().getFirst())

if ((a->getContent().getFirst())->getSize().empty())

deepness(dynamic\_cast<Directory\*>(a->getContent().getFirst()), i, r, ir, NULL);

else

deepness(NULL, i, r, ir, dynamic\_cast<File\*>(a->getContent().getFirst()));

if (el && el->next\_elem())

if (!((el->next\_elem()->getSize().empty())))

deepness(NULL, i, r, ir, dynamic\_cast<File\*>(el->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(el->next\_elem()), i, r, ir, NULL);

if (a && a -> next\_elem())

if (!((a->next\_elem()->getSize().empty())))

deepness(NULL, i, r, ir, dynamic\_cast<File\*>(a->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(a->next\_elem()), i, r, ir, NULL);

if (el == NULL)

{

if (a->getParam().attributes.invisible && a->getParam().attributes.readOnly)

ir++;

else

if (a->getParam().attributes.invisible)

i++;

else

if (a->getParam().attributes.readOnly)

r++;

}

if (a == NULL)

{

if (el->getParam().attributes.invisible && el->getParam().attributes.readOnly)

ir++;

else

if (el->getParam().attributes.invisible)

i++;

else

if (el->getParam().attributes.readOnly)

r++;

}

}

};

class DVDDirectoryCheck : public DirectoryCheck

{

public:

DVDDirectoryCheck(Directory \*toCheck) : DirectoryCheck(toCheck) {}

void summarize()

{

bool deep = true;

cout << "\nChecking direcory " << toCheck->getStr() << "...";

deepness(toCheck, deep, NULL);

if (deep)

cout << "\nCatalogue " << toCheck->getStr() << " can be written on DVD\n";

else

cout << "\nCatalogue " << toCheck->getStr() << " cannot be written on DVD\n";

}

void deepness(Directory\* a, bool &b, File\* el)

{

if (el == NULL)

if (a->getContent().getFirst())

if ((a->getContent().getFirst())->getSize().empty())

deepness(dynamic\_cast<Directory\*>(a->getContent().getFirst()), b, NULL);

else

deepness(NULL, b, dynamic\_cast<File\*>(a->getContent().getFirst()));

if (el && el->next\_elem())

if (!((el->next\_elem()->getSize().empty())))

deepness(NULL, b, dynamic\_cast<File\*>(el->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(el->next\_elem()), b, NULL);

if (a && a -> next\_elem())

if (!((a->next\_elem()->getSize().empty())))

deepness(NULL, b, dynamic\_cast<File\*>(a->next\_elem()));

else

deepness(dynamic\_cast<Directory\*>(a->next\_elem()), b, NULL);

if (el == NULL)

{

if (a->getParam().depth > toCheck->getParam().depth + 8)

b = false;

if (DVDFileCheck(new File(a->getParam())).summarize(true))

b = false;

}

else

{

if (el->getParam().depth > toCheck->getParam().depth + 8)

b = false;

if (DVDFileCheck(new File(el->getParam())).summarize())

b = false;

}

}

};

string WindowsFileCheck::restrictedNames[4] = {"con", "nul", "prn", "aux"};

char WindowsFileCheck::restrictedSymbols[10] = {'\0', '/', '\\',

':', '\*', '?', '"', '<' , '>', '|'};

# Приложение А: длинная арифметика

## digit.h

#pragma once

class Digit

{

public:

virtual void out() = 0;

virtual Digit \* myCopy() = 0;

virtual bool operator!=(Digit&) = 0;

virtual bool operator>(Digit&) = 0;

virtual Digit \*sum(Digit \*) = 0;

virtual Digit \*sub(Digit \*) = 0;

virtual Digit \*mult(Digit \*) = 0;

virtual Digit \*normalize() = 0; //0<digit<base

virtual Digit \*zero() = 0; //digit = 0, return digit

};

## digit10.h

#include "digit.h"

#include "iostream"

using namespace std;

class Digit10 : public Digit

{

int a;

public:

Digit10() : a(0) {};

Digit10(char x) : a(x) {};

void out() {cout<<(int)a;}

Digit \* myCopy();

bool operator!=(Digit&);

bool operator>(Digit&);

Digit \* sum(Digit\*);

Digit \* sub(Digit\*);

Digit \* mult(Digit\*);

Digit \* normalize();

Digit \* zero() {a = 0; return this;}

};

Digit \* Digit10::myCopy()

{

Digit \*ans = new Digit10(a);

return ans;

}

bool Digit10::operator!=(Digit &b)

{

Digit10 ex = dynamic\_cast<Digit10 &> (b);

return a!=ex.a;

}

bool Digit10::operator>(Digit &b)

{

Digit10 ex = dynamic\_cast<Digit10 &> (b);

return a>ex.a;

}

Digit \* Digit10::sum(Digit \*b)

{

Digit10 beta = dynamic\_cast<Digit10&>(\*b);

Digit \*ex = new Digit10(a + beta.a);

return ex;

}

Digit \* Digit10::sub(Digit \*b)

{

Digit10 beta = dynamic\_cast<Digit10 &>(\*b);

Digit \*ex = new Digit10(a - beta.a);

return ex;

}

Digit \* Digit10::mult(Digit \*b)

{

Digit10 beta = dynamic\_cast<Digit10 &>(\*b);

Digit \*ex = new Digit10(a \* beta.a);

return ex;

}

Digit \* Digit10::normalize()

{

Digit \*ans;

if (a>=0)

{

ans = new Digit10(a/10);

a%=10;

}

else

{

ans = new Digit10(-1);

a+=10;

}

return ans;

}

## Long.h

#pragma once

#include "digit.h"

#include "digit10.h"

#include <iostream>

using namespace std;

class Long

{

Digit \*\*a;

int length;

bool isNegative;

public:

Long() : a(NULL), length(0), isNegative(false) {};

~Long()

{

if (a) return;

for (int i = 0; i<length; i++)

delete a[i];

delete [] a;

}

void setNum(Digit \*\*, int, bool);

void outNum();

Long involution(long);

Long& operator=(const Long &);

friend bool operator==(const Long&, const Long&);

friend bool operator!=(const Long&, const Long&);

friend bool operator>(const Long&, const Long&);

friend bool operator<(const Long&, const Long&);

friend bool operator<=(const Long&, const Long&);

friend bool operator>=(const Long&, const Long&);

friend Long& operator-(const Long&, const Long&);

friend bool operator>(const Long&, string);

friend Long& operator\*(const Long&, const Long&);

friend Long& operator\*(const Long&, string);

friend Long& operator+(const Long&, const Long&);

friend Long& operator-(const Long &a)

{

Long \*ex = new Long;

ex->a = new Digit\*[a.length];

ex->length = a.length;

for (int i = 0; i < ex->length; i++)

ex->a[i] = a.a[i]->myCopy();

ex->isNegative = !a.isNegative;

return \*ex;

}

friend Long& operator+(const Long &a)

{

Long \*ex = new Long;

ex->a = new Digit\*[a.length];

ex->length = a.length;

for (int i = 0; i < ex->length; i++)

ex->a[i] = a.a[i]->myCopy();

ex->isNegative = a.isNegative;

return \*ex;

}

bool empty();

};

Long& operator+(const Long &a, const Long &b)

{

Long \*ex = new Long;

if (a.isNegative && !b.isNegative || !a.isNegative && b.isNegative)

{

if (a.isNegative)

return b-(-a);

else

return a-(-b);

}

if (a.length<b.length)

return b + a;

ex->a = new Digit\*[b.length + 1];

Digit \*temp;

Digit \*carry = a.a[0]->normalize();// Carry = 0 Т\_Т

int i;

for (i = 0; i<b.length; i++)

{

temp = (a.a[i]->sum(b.a[i]))->sum(carry); // O\_O temp = a + b + carry

carry = temp->normalize();

ex->a[i] = temp;

}

for (;i<a.length; i++)

{

temp = a.a[i]->sum(carry);

carry = temp->normalize();

ex->a[i] = temp;

}

Digit \*z = carry->myCopy()->zero();

if (\*carry != \*z)

{

ex->a[i] = carry;

ex->length = a.length + 1;

}

else

ex->length = a.length;

ex->isNegative = a.isNegative;

return \*ex;

}

Long& operator-(const Long &a, const Long &b)

{

if (a.isNegative && !b.isNegative || !a.isNegative && b.isNegative)

{

if (a.isNegative) return -(-a+b);

else return a+(-b);

}

if (a.length < b.length)

return -(b-a);

Long \*ex = new Long;

ex->a = new Digit\*[b.length + 1];

Digit \*temp;

Digit \*carry = a.a[0]->normalize();// Carry = 0 Т\_Т

int i;

for (i = 0; i<b.length; i++)

{

temp = a.a[i]->sub(b.a[i])->sum(carry);

carry = temp->normalize();

ex->a[i] = temp;

}

for (; i<a.length; i++)

{

temp = a.a[i]->sum(carry);

carry = temp->normalize();

ex->a[i] = temp;

}

i = a.length - 1;

Digit \*z = carry->myCopy()->zero();

while ( i>0 && !(\*ex->a[i]!=\*z)) i--;

ex->length = i + 1;

ex->isNegative = (a>=b)? false : true;

return \*ex;

}

bool operator!=(const Long &a, const Long &b)

{

return !(a==b);

}

Long Long::involution(long k)

{

Long a(\*this);

for (int i = 0; i < k - 1; i++)

\*this = (\*this) \* a;

return \*this;

}

Long assign(Long &l, string b)

{

Digit\*\* a = new Digit\*[b.length()];

for (int i = 0; i < b.length(); i++)

a[i] = new Digit10(b[i]-'0');

l.setNum(a, b.length(), false);

return l;

}

bool Long::empty()

{

Long a;

return (\*this == assign(a, "0"));

}

Long& operator\*(const Long& a, string b)

{

Long t;

assign(t, b);

return a\*t;

}

void Long::setNum(Digit \*\*num, int length, bool isNegative)

{

this->isNegative = isNegative;

this->length = length;

a = new Digit\*[length];

for (int i = length - 1; i>=0; i--)

a[i] = num[length - i - 1]->myCopy();

}

void Long::outNum()

{

if (isNegative) cout<<'-';

for (int i = length - 1; i>=0; i--)

a[i]->out();

}

Long& Long::operator=(const Long &sec)

{

isNegative = sec.isNegative;

length = sec.length;

a = new Digit\*[length];

for (int i = 0; i<length; i++)

a[i] = sec.a[i]->myCopy();

return \*this;

}

bool operator>(const Long& a, string b)

{

Long c;

assign(c, b);

return a > c;

}

bool operator>(const Long &a, const Long &b)

{

bool flag = false;

if (a.isNegative && !b.isNegative || a.length<b.length) return false;

if (!a.isNegative && b.isNegative || a.length>b.length) return true;

for (int i = 0; i < a.length; i++)

if (\*a.a[i]>\*b.a[i])

{

flag = true;

break;

}

if (a.isNegative)

return !flag;

else

return flag;

}

Long& operator\*(const Long &a, const Long &b)

{

if (a.length<b.length)

return b\*a;

Long \*ex = new Long;

Digit \*temp;

Digit \*carry = a.a[0]->normalize();// Carry = 0 Т\_Т

Digit \*zero = carry->myCopy();//Ноль

ex->a = new Digit\*[a.length + b.length];

for (int i = 0; i<a.length + b.length+1; i++)

ex->a[i] = zero->myCopy(); // обнуление

int i,j;

for (i = 0; i<a.length; i++)

{

carry = zero->myCopy();

for (j = 0; j<b.length; j++)

{

temp = ((a.a[i]->mult(b.a[j]))->sum(ex->a[i+j]))->sum(carry);

carry = temp->normalize();

ex->a[i+j] = temp;

}

ex->a[i+j] = carry;

}

i = a.length + b.length - 1;

if (!(\*ex->a[i] != \*zero)) i--;

ex->length = i + 1;

ex->isNegative = a.isNegative && !b.isNegative || !a.isNegative && b.isNegative;

return \*ex;

}

bool operator==(const Long &a, const Long &b)

{

bool flag = false;

if (a.length == b.length && a.isNegative == b.isNegative)

{

flag = true;

for (int i = 0; i<a.length; i++)

if (\*a.a[i] != \*b.a[i])

{

flag = false;

break;

}

}

return flag;

}

bool operator<(const Long &a, const Long &b)

{

if (a>b || a==b) return false;

else return true;

}

bool operator<=(const Long &a , const Long &b)

{

return a<b || a==b;

}

bool operator>=(const Long &a , const Long &b)

{

return a>b || a==b;

}

# Приложение B: Тестирующий код программы main.cpp

#include "FileSystem.h"

int main()

{

setlocale(LC\_ALL, "rus");

Long aa;

Directory a(assign(aa, "0"), "ROOT", 0, 1);

Directory\* push;

a.push\_back(push = new Directory(assign(aa, "0"),"D1",0,0));

Directory\* save;

File\* saveF1, \*saveF2;

push -> push\_back(save = new Directory(assign(aa, "0"), "D2", 0, 0));

save -> push\_back(new File(assign(aa, "1231"), "F1", 0, 1))

.push\_back(new File(assign(aa, "11"), "F2", 0, 0));

push -> push\_back(4, new File(assign(aa, "12"), "F2", 1, 0),

saveF1 = new File(assign(aa, "2").involution(2),"F3", 1, 1),

saveF2 = new File(assign(aa, "129"),"F4", 0, 0),

new File(assign(aa, "12"),"F5", 0, 0));

a.push\_back(new Directory(assign(aa, "0"),"D3",0,0));

a.prepare();

cout << a.str();

WindowsFileCheck(saveF1).summarize();

DVDFileCheck dvd(saveF1);

dvd.summarize();

dvd = saveF2;

dvd.summarize();

DVDDirectoryCheck dvdD(&a);

dvdD.summarize();

AttributeCheck attrCheck(&a);

attrCheck.summarize();

cout << endl;

system("pause");

}

# Литература

<http://ru.wikipedia.org/wiki/NTFS>

<http://ru.wikipedia.org/wiki/Ext>

<http://ru.wikipedia.org/wiki/FAT32>

<http://ru.wikipedia.org/wiki/DVD>

Шилдт Г. - Полный справочник по C++ (4-е издание)

Lubos Rendek, перевод: А.Кривошей, "Типы файлов в Linux"