Міністэртсва адукацыі Рэспублікі Беларусь

Установа адукацыі

«Беларускі дзяржаўны ўніверсітэт

інфарматыкі і радыёэлектронікі»

Факультэт камп’ютарных сетак і сістэм

Кафедра электронных вылічальных машын

Дысцыпліна: Праграмаванне на мовах высокага ўзроўня

ТЛУМАЧАЛЬНАЯ ЗАПІСКА

да курсавога праекта

на тэму

ПРАВЕРКА ЦЭЛАСНАСЦІ ФАЙЛАВАЙ СІСТЭМЫ FAT32

БДУІР КП 1-40 02 01 002 ТЗ

Студэнт Я. Д. Бекетаў

Кіраўнік Д. У. Басак

МІНСК 2023

**ЗМЕСТ**

УВОДЗІНЫ 5

1. АГЛЯД ЛІТАРАТУРЫ 6

2. СТРУКТУРНАЕ ПРАЕКТАВАННЕ 12

3. ФУНКЦЫЯНАЛЬНАЕ ПРАЕКТАВАННЕ 16

4. ІНСТРУКЦЫЯ КАРЫСТАЛЬНІКА 28

ЗАКЛЮЧЭННЕ 29

СПІС ВЫКАРЫСТАВАННЫХ КРЫНІЦ 30

ПРЫДАТАК А 31

ПРЫДАТАК Б 80

ПРЫДАТАК В 83

ПРЫДАТАК Г 84

ПРЫДАТАК Д 85

Міністэртсва адукацыі Рэспублікі Беларусь

Установа адукацыі

«Беларускі дзяржаўны ўніверсітэт

інфарматыкі і радыёэлектронікі»

Факультэт кампьютарных сетак і сістэм

СЦВЯРДЖАЮ

Загадчык кафедры Нікульшын Б. В.

*––––––––––––––––––––––––*

(подпіс)

–––––––––––––––––––2023 г.

ЗАДАННЕ

па курсавому праектаванню

Студэнту    *Бекетаву Яўгенію Дзмітрыевічу–––––––––––––––––––*

1. Тэма праекта   Праверка цэласнасці файлавай сістэмы FAT32         ––   –– –

2. Тэрмін здачы студэнтам скончанага праекта–––––*15 снежня 2022 г.–—––*

3. Змест разлікова-тлумачальнай запісцы (спіс пытанняў, каторыя падлягаюць распрацоўцы)

*Уводзіны. 1. Агляд літаратуры. 2. Структурнае праектаванне. 3. Функцыянальнае праектванне. 4. Інструкцыя карыстальніка. Заключэнне. ПрыдатакА. Прыдатак Б. Прыдатак В. Прыдатак Г. Прыдатак Д.*

5. Спіс графічнага матэрыяла:

*1. Скрыншоты працы праграмы –––––––––––––––––––––––––––––*

*2. UML-дыяграма класаў. –––––––––––––––––––––––––––––*

*3. Схема асноўных алгарытмаў праграмы. –––––––––––––––––––––––––– ––*

*4. Ведамасць дакументаў. . –––––––––––––––––––––––– –––––*

6. Кансультант па праекту   *Д. У. Басак ––––––––––––––––––––––––– –––––*

7. Дата выдачы задання –––––*16 верасня 2022 г. ––––––––––––––––––––– –*

8. Календарны графік працы над праектам на ўвесь перыяд праектавання (з вызначэннем тэрмінаў выканання і працаёмкасць асобных этапаў):

*раздзел 1 да 15 10 – 25 %;––––––––––––––––––––––––––––––––––––––*

*раздзелы 2, 3 да 15 11 – 40 %;–––––––––––––––––––––––––––––––––––––*

*раздзел 4 да 15 12 – 15 %;––––––––––––––––––––––––––––––––––––––––*

*афармденне тлумачальнай запісцы і графічнага матэрыяла да 15 12 – 20 %*

*Абарона курсавога праекта з 19 па 30 снежня––––––––––––––––––––––––––––*

КІРАЎНІК*– ––––––––––– Д. У. Басак*

(подпіс)

Заданне прыняў да выканання*–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_–– Я. Д. Бекетаў*

(дата і подпіс студэнта)

**УВОДЗІНЫ**

Файлавыя сістэмы – гэта метад і структура дадзеных, з дапамогай каторай камп’ютар кантралюе і кіруе запісанымі дадзеннымі.

У 1977 годзе кампанія MICROSOFT распрацавала новую файлавую сістэму для флопі-дыскаў, каторую назвалі “FAT”(File Allocation Table). Яе наступныя версіі пачалі выкарыстоўваць ужо ў пазнейшых аперацыйных сістэмах DOS і ранейшых Windows. На цяперашні час FAT усталёюваць на партаціўныя прылады, мабільныя тэлефоны, флэш-дыскі. FAT падыходзіць да вялікай колькасці аперыйных сістэм(Windows, MacOs, Linux).

Мэтай курсавога праекта з’яўляецца напісанне праграмы праверкі цэласнасці файлавай сістэмы FAT32 з дапамогай атрыманых ведаў у ходзе вывучэння дысцыпліны “Аперацыйныя сістэмы і сістэмнае праграмаванне” і тэматычнай літаратуры.

Дадзенная праграма рэалізаваная на мове праграмавання “С” для аперацыйнай сістэмы Linux. На выбар такой рэалізацыі паўплывала нізка-ўзроўневасць “C” і прамы доступ да ядра Linux, што значна спрасціла напісанне курсавога праекта і адначасова ўдасканаліла веды ў працы аперацыйнай сістэмы.

Для дасягнення мэты былі пастаўленыя наступныя задачы: вывучыць неабходную інфармацыю па структуры FAT і стварыць кансольную праграму.

**1 АГЛЯД ЛІТАРАТУРЫ**

FAT файлавая сістэма выкарыстоўвае індэксныя табліцы, каб вызначаць месцы ў фізічнай памяці, каторыя выкарыстоўваюць файлы сістэмы. Пад FAT вызначаецца асаблівае месца ў памяці пад час фармаціравання. Файлавая табліца(FAT) – гэта звязаны спіс запісаў для кожнага кластара(неперапыннай вобласці дыскавага сховішча). Запіс можа змяшчаць рад значэнняў: нумар наступнага кластара ў ланцужку, EOC(канец ланцужка), зарэзерваваны кластар, дрэнны кластар і вольны кластар. Існуе некалькі версій файлавай сістэмы FAT: FAT12, FAT16, FAT32. Кожная версія мала свае асаблівасці. Адно з асноўных: колькасць бітов, што выкарыстоўваецца для запісаў у табліцу. У гэтым курсавым праекце разглядаецца FAT32.

FAT32 складаецца з трох раздзелаў: Зарэзерваваныя сектары, FAT раздзел і раздзел дадзеных.

**1.1 Зарэзерваваныя сектары**

**1.1.1 Сектар загрузкі**

Сектар загрузкі – гэта першы лагічны сектар, каторы складаецца з BIOS Parameter Block(BPB) і Extend BPB і змяшчае базавую інфармацыю па файлавай сістэме(FSInfo).

Структура BIOS Parameter Block:

|  |  |  |  |
| --- | --- | --- | --- |
| Змяшчэнне  (дзесятковае) | Змяшчэнне  (шаснацтковая) | Памер, байты | Назначэнне |
| 0 | 0x00 | 3 | Змяшчае байты EB 3C 90, каторыя роўныя асэмблернаму коду JMP SHORT 3C NOP(3С можа адрознівацца). Выкарыстоўваецца для таго, каб “пераскочыць” праз BPB і EBPB. |
| 3 | 0x03 | 8 | Назва выкарыстоўваемай аперацыйнай сістэмы |
| 11 | 0x0B | 2 | Памер сектара ў байтах. |
| 13 | 0x0D | 1 | Колькасць сектараў у кластары. |
| 14 | 0x0E | 2 | Колькасць зарэзерваваных сектараў з улікам сектара запуску. |
| 16 | 0x10 | 1 | Колькасць файлавых табліц. |
| 17 | 0x11 | 2 | Колькасць запісаў каранёвага каталога. |
| 19 | 0x13 | 2 | Агульная колькасць сектараў на дыску. Калі гэтае значэнне роўна 0, тады рэальная колькасць сектараў большая за 65535. Дакладнае значэнне ляжыць па змяшчэнню 0х20. |
| 21 | 0x15 | 1 | Вызначае тып мэдыа дэскрыптара. |
| 22 | 0x16 | 2 | Колькасць сектараў на табліцу. Толькі для FAT12 і FAT16. |
| 24 | 0x18 | 2 | Колькасць сектараў на трэк. |
| 26 | 0x1A | 2 | Колькасць запісваючых галовак. |
| 28 | 0x1C | 4 | Колькасць схаваных сектараў. |
| 30 | 0x20 | 4 | Вялізная колькасць сектараў. |

Табліца 1.1.1.1 – структура BIOS Parameter Block.

Структура Extended BPB:

|  |  |  |  |
| --- | --- | --- | --- |
| Змяшчэнне  (дзесятковае) | Змяшчэнне  (шаснацтковая) | Памер, байты | Назначэнне |
| 36 | 0x024 | 4 | Памер файлавай табліцы(у сектарах). |
| 40 | 0x028 | 2 | Сцягі. |
| 42 | 0x02A | 2 | Версія файлавай сістэмы. |
| 44 | 0x02C | 4 | Нумар першага кластара каранёвага каталога. Звычайна роўна 2. |
| 48 | 0x030 | 2 | Нумар сектара FSInfo структуры. |
| 50 | 0x032 | 2 | Нумар сектара копіі “сектара загрузкі”. |
| 52 | 0x034 | 12 | Зарэзерваваныя. Калі дыск адфарматаваны, гэтыя байты павінны быць роўнымі нулю. |
| 64 | 0x040 | 1 | Нумар дыска. |
| 65 | 0x041 | 1 | Сцягі Windows NT. |
| 66 | 0x042 | 1 | Подпіс (0x28 альбо 0x29). |
| 67 | 0x043 | 4 | ID дыска. |
| 71 | 0x047 | 11 | Назва дыска. |
| 82 | 0x052 | 8 | Назва файлавай сістэмы. Заўжды "FAT32 ". |
| 90 | 0x05A | 420 | Код запуску. |
| 510 | 0x1FE | 2 | Подпіс загрузнага падзела(partition) 0xAA55. |

Табліца 1.1.1.2 – структура Extended BPB.

**1.1.2 FS Information сектар**

Структура FSInfo:

|  |  |  |  |
| --- | --- | --- | --- |
| Змяшчэнне  (дзесятковае) | Змяшчэнне  (шаснацтковая) | Памер, байты | Назначэнне |
| 0 | 0x0 | 4 | Вядучы подпіс 0x41615252. |
| 4 | 0x4 | 480 | Зарэзерваваныя. |
| 484 | 0x1E4 | 4 | Подпіс(0x61417272). |
| 488 | 0x1E8 | 4 | Колькасць вольных кластараў. Калі значэнне – 0xFFFFFFFF, тады колькасць вольных кластараў невядомая і павінна быць падлічана. |
| 492 | 0x1EC | 4 | Indicates the cluster number at which the filesystem driver should start looking for available clusters. If the value is 0xFFFFFFFF, then there is no hint and the driver should start searching at 2. Typically this value is set to the last allocated cluster number. As the previous field, this value should be range checked. |
| 496 | 0x1F0 | 12 | Зарэзерваваныя. |
| 508 | 0x1FC | 4 | Подпіс (0xAA550000) |

Табліца 1.1.2.1 – структура FSInfo сектара.

**1.2 FAT раздзел**

У гэтым раздзеле знаходзяцца файлавыя табліцы(звычайна дзве, прычым другая з’яўляецца копіяй першай). З дапамогай гэтых табліц вызначаецца наступны раздзел дадзенных. Табліца складаецца з запісаў, памер каторых вызначаецца ў залежнасці ад версіі FAT. У выпадку FAT32 выкарыстоўваецца 32 біта, 4 з каторых зарэзерваваныя. Пры фармаціраванні дыска зарэзерваваныя біты звычайна выстаўляюцца ў нуль.

Значэнні запісаў у файлавай табліцы:

|  |  |
| --- | --- |
| Значэнне запіса | Назначэнне |
| 0х?0000000 | Вольны кластар. Калі такое значэнне з’яўляецца ў ланцужку кластараў, яно павінна ўспрымацца як EOC. |
| 0x?0000001 | Зарэзерваваны для ўнутраных мэт. Калі такое значэнне з’яўляецца ў ланцужку кластараў, яно павінна ўспрымацца як EOC. |
| 0x?0000002 - 0x?FFFFFEF | Нумар наступнага кластара ў ланцужку. |
| 0x?FFFFFF0 - 0x?FFFFFF5 | Зарэзерваваны кластар. Калі з’яўляецца ў ланцужках кластараў, павінна ўспрымацца як нумар наступнага кластара. |
| 0x?FFFFFF6 | Зарэзерваваны. |
| 0x?FFFFFF7 | Дрэнны кластар. |
| 0x?FFFFFF8 - 0x?FFFFFFF | Апошні кластар у ланцужку. Значэнне 0x0FFFFFF8 выкарыстоўваецца mkdosfs для каранёвых каталогаў. |

Табліца 1.2.1 – Значэнні запісаў файлавай табліцы.

**1.3 Раздзел дадзеных**

Гэты раздзел змяшчае табліцу каранёвага каталога і табліцы астатніх каталогаў. Табліца каталогаў складаецца з 32-байтных запісаў(entry).  
 Структура запісу каталога:

|  |  |  |
| --- | --- | --- |
| Змяшчэнне(у байтах) | Памер  (у байтах) | Назначэнне |
| 0х00 | 8 | Скарочанае файлавае імя. У першым байце могуць знаходзіцца сцягі:  1. 0х2Е – маркіруе запісы “.” і “..”;  2. 0х5Е – маркіруе файл як выдалены. |
| 0x08 | 3 | Скарочнае расшырэнне файла. |
| 0x0В | 1 | Файлавыя атрыбуты:  1. 0х01 – толькі для чытання;  2. 0х02 – схаваны файл;  3. 0х04 – сістэмны файл (не можа быць выдалены карыстальнікам);  4. 0х08 – назва сховішча;  5. 0х10 – каталог;  6. 0х20 – архіў. |
| 0x0C | 1 | Зарэзерваваны. |
| 0x0D | 1 | Дзясятыя секунды стварэння файла. |
| 0x0Е | 2 | Час стварэння. Час – 15 - 11 біты, хвіліны – 10 - 5 біты, секунды – 4 - 0 біты. |
| 0x10 | 2 | Дата стварэння. Год – 15 - 9 біты, месяц – 8 - 5 біты, дзень – 4 - 0 біты. |
| 0х12 | 2 | Дата апошняга доступу (па структуры як 0х10). |
| 0х14 | 2 | Два верхніх байта нумара кластара, па катораму ляжаць дадзеныя файла. |
| 0х16 | 2 | Час апошняга змянення (па структуры як 0х0Е). |
| 0х18 | 2 | Дата апошняга змянення (па структуры як 0х10). |
| 0х1А | 2 | Два ніжніх байта нумара кластара, па катораму ляжаць дадзеныя файла. |
| 0х1С | 4 | Памер файла ў байтах. |

Табліца 1.3.1 - Структура запісу каталога.

Перад кожным запісам каталога звычайна змяшчаецца запіс з дліннай файлавай назвай (long file name entry).

Структура такога запісу выглядае такім чынам:

|  |  |  |
| --- | --- | --- |
| Змяшчэнне(у байтах) | Памер  (у байтах) | Назначэнне |
| 0х00 | 1 | Вызначае парадак запісаў з дліннай файлавай назвай. |
| 0x01 | 10 | Першыя пяць 2-х байтных сімвалаў назвы. |
| 0x0B | 1 | Файлавыя атрыбут – 0x0F. |
| 0x0C | 1 | 0x00. |
| 0x0D | 1 | Чэк-сума кароткай назвы файла. |
| 0x0Е | 12 | Наступныя шэсць 2-х байтных сімвалаў назвы. |
| 0x1А | 2 | 0х0000. |
| 0х1С | 4 | Апошнія два 2-х байтных сімвалаў назвы. |

Табліца 1.3.2 – Структура запісу з дліннай файлавай назвай.

**1.4** **Праверка цэласнасці файлавай сістэмы**

Для таго каб ацаніць карэктнасць файлавай сістэмы FAT32, трэба напісаць праграму здольную прасканіраваць сістэму. Пад час такога сканіравання можна праверыць цэласнасць файлавай сістэмы.

Праблемы і памылкі, што павінна вызначаць праграма праверкі цэласнасці файлавай сістэмы:

1. У табліцы знаходзіцца некарэктны нумар кластара;
2. Зацыклены ланцужок файлаў;
3. Дрэнныя кластары;
4. Дрэнныя і пашкоджаныя запісы каталогу;
5. Файлы, што не з’яўляюцца каталогам, з назвай «.» і «..»;
6. Каталогі з не нулявым значэннем памера файла;
7. Каталог «.» не спасылаецца на бацькоўскі каталог;
8. Каталог «..» не спасылаецца на бацькоўскі каталог бацькоўскага каталога;
9. У запісу файла некарэктны кластар;
10. Несупадзенне памераў файла з выкарыстоўваемымі кластарамі;
11. Памылкі ў раздзеле загрузкі і структуры FSInfo.

У дадзеным курсавым праекце праграма праверцы цэласнасці файлававай сістэмы выводзіць не толькі памылкі, але і інфармацыю пра зарэзерваваныя сектары і файлы дыска(у залежнасці ад таго, якія опцыі выбраў карыстальнік пры запуску праграмы).

**2 СТРУКТУРНАЕ ПРАЕКТАВАННЕ**

Праграмны код быў разбіты на модулі для больш зручнага карыстання і змянення функцыянала праграмы.

* 1. **Файл source.c**

**1.1.1 int main(int argc, char\*\* argv)**

У main-функцыі адбываецца першае кантактаванне з карыстальнікам. Main прымае аргументы каманднага радка, і каб прааналізаваць іх вызывае int check\_args(int argc, char\*\* argv). Калі функцыя check\_args не знайшла памылак, вызываецца int myfsck(int argc, char\*\* argv), з каторай пачынаецца праверка файлавай сістэмы дыска.

**1.1.2 int check\_args(int argc, char\*\* argv)**

Пры выяўленні дадзенай функцыяй некарэктных значэнняў аргументаў каманднага радка, вызываецца функцыя print\_usage(). Для праверкі опцый вызначаных карыстальнікам вызываецца фукнцыя int get\_options(int argc, char\*\* argv), а для праверкі уведзенай назвы дыска – бібліятэчная функцыя int open(const char \*\_\_file, int \_\_oflag, ...).

**1.1.3 int get\_options(int argc, char\*\* argv)**

З дапамогай бібліятэчнай функцыі getopt счытваюцца і захоўваюцца значэнні ўведзенных опцый. Калі опцый не было, альбо былі ўведзены няправільна, пераменная options застаецца роўнай 16.

**1.1.4** **void print\_usage()**

Дадзеная функцыя выводзіць у тэрмінал даведку, як карыстацца праграмай.

**1.1.5 int myfsck(int argc, char\*\* argv)**

З гэтага метада пачынаецца выклікасноўных функцый для праверкі цэласнасці файлавай сістэмы: open(для адкрыцца правяраемага дыску), read\_boot(для чытання раздзела загрузкі), read\_fat(для чытання fat раздзела), read\_disk(для чытання дыску).

**1.2 Файл source.h**

У гэтым хэдары знаходзяцца пералічэнні для вызначэння стана правяраемай часткі файлавай сістэмы(enum TEST\_STATE), для выкарыстоўваемых опцый(enum OPTIONS) і для вызначэння тыпу файла(enum FILE\_ATTR), дэфайны для агульнай працы праграмы і прататыпы глабальных пераменных з файла global\_vars.с.

**1.3 Файл fat32.c**

**1.3.1 void fs\_read(off\_t offset, int size, void\* data)**

Простая функцыя для счытвання інфармацыі з файлавай сістэмы.

**1.3.2 void get\_fat(struct FAT\_ENTRY\* entry, void\* fat,**

**uint32\_t cluster)**

Дадзеная функцыя «дастае» запіс з файлавай табліцы па вызначаннаму кластару.

**1.3.3 uint32\_t get\_entry\_start(struct DIR\_ENT\* entry)**

Вяртае нумар кластара, што выкарыстоўвае прачытаны файл.

**1.3.4 off\_t base\_cluster(uint32\_t cluster)**

Функцыя вяртае змяшчэнне ў байтах па кластару ўнутры раздзела дадзеных.

**1.3.5 int read\_boot()**

Дадзеная функцыя счытвае значэнне раздзела загрузкі, пры вызначаных опцыях выводзіць значэнні гэтага раздзела, правярае гэтыя значэнні на наяўнасць памылак. Для вывада значэнняў раздзела выкарыстоўвае функцыю pretty\_output.

**1.3.6 int init\_file\_sys()**

Асноўнае назначэнне гэтай функцыі – ініцыалізацыя глабальнай пераменнай file\_sys, што змяшчае важную інфармацыю пра файлавую сістэму дыска. Паралельна з запісам асноўных значэнняў адбываецца іх праверка.

**1.3.7 int read\_fsinfo()**

Функцыя read\_fsinfo выкарыстоўваецца для счытвання і праверцы fsinfo сектара. Пры вызначаных опцыях выводзіць дадзеныя сектара.

**1.3.8 int read\_fat()**

Функцыя счытвае адну ці дзве(калі існуе) файлавыя табліцы і правярае іх на памылкі.

**1.3.9 void read\_disk(uint32\_t cluster, uint32\_t grandp, char\* file\_path)**

Рэкурсіўная функцыя read\_disk ажыццяўляе асноўную праверку файлавай сістэмы: сканіруе ўсі файлы сістэмы, правярае назвы файлаў, запісы файлаў і цыклы ў кластаравым ланцужку. Для праверцы файлаў read\_disk вызывае функцыю int test\_file(uint8\_t isroot, struct DOS\_FILE\* file).

**1.3.10 void pretty\_output()**

Вывад раздзела загрузкі.

**1.4 Файл fat32.h**

У гэтай файле змяшчаюцца прататыпы функцый з fat32.c.

**1.5 Файл check\_fs.h**

**1.5.1 int test\_file(uint8\_t isroot, struct DOS\_FILE\* file)**

Дадзенная функцыя праводзіць поўную праверку файла. На першым этапе правяраюцца файлавае імя і атрыбуты. Калі атрыбут файл «VOLUME\_ID» альбо файлавае імя пачынаецца са сцяга «0хЕ5» праверка файла скончваецца. У астатніх выпадках з дапамогай функцыі int check\_file\_name(unsigned char\* name, uint8\_t isroot, uint8\_t attr) правяраецца імя. Калі імя адпавядае ўмовам добрай файлавай назвы, вызываецца функцыя void change\_file\_name(char\* name, uint8\_t attr), што змяняе імя файла для больш зручнага чытання пры яго вывадзе ў тэрмінал, з дапамогай int output\_file\_info(struct DOS\_FILE\* file, unsigned char\* file\_name, uint8\_t attr). На наступных этапах выканання функцыі test\_file праверка каталогаў і звычайных файлаў адрозніваецца. Для каталогаў правяраецца каталожны запіс. А для звычайнага файла, акрамя каталожнага запісу, правяраецца на карэктнасць выкарыстоўваемыя кластары, іх сумарны памер і здольнасць счытаць інфармацыю з вобласці дыска, на каторую яны ўказваюць.

**1.5.2 int check\_file\_name(unsigned char\* name, uint8\_t isroot,**

**uint8\_t attr)**

Як узгадвалася вышэй, гэтая функцыя выкарыстоўваецца для праверкі імя файла.

Функцыя вызначае, што назвы файла «дрэнная», у наступных выпадках:

* Калі першы сімвал імя прабел;
* Калі файлавае імя пачынаецца з кропкі і файл знаходзіцца ў каранёвым каталогу, альбо з'яўляецца звычайным.
* Пры наяўнасць «дрэнных» сімвалаў:"\*?/\\|,;:+=<>[]\"".
* Калі ў імені ёсць прабелы паміж сімваламі.

**1.5.3 void change\_file\_name(char\* name, uint8\_t attr)**

Дадзеная функцыя мяняе файлавае імя для больш зручнага вывада ў тэрмінал. З імя прыбіраюцца лішнія прабелы і дабаўляецца кропка перад расшырэннем файла.

**1.5.4 int output\_file\_info(struct DOS\_FILE\* file,**

**unsigned char\* file\_name, uint8\_t attr)**

Калі карыстальнік абраў пэўныя опцыі, што дазваляюць вывад, гэта функцыя выводзіць інфармацыю пра файл і дабаўляе імя файла да пераменнай file->path. У іншым выпадку толькі змяняе file->path.

**1.5.5 int check\_fat(void\* fat, struct FAT\_ENTRY\* fat\_media,**

**off\_t offset, int size)**

Функцыя check\_fat счытвае запіс з файлавай табліцы і правярае яго на карэктнасць.

**1.5.6 int fs\_check(off\_t pos, int size)**

Гэтая функцыя вызначае ці магчыма счытаць інфармацыю з дыска па зададзенаму змяшчэнню і памеру.

**1.5.7 uint32\_t next\_cluster(uint32\_t cluster)**

Вяртае значэнне наступнага кластара ў ланцужку.

**1.6 Файл check\_fs.h**

Змяшчае пратыпы функцый файла check\_fs.c.

**1.7 Файл global\_vars.c**

У гэтым файле пералічаны наступныя глабальныя пераменныя:

* fd – адказвае за змяшчэнне файлавага дэскрыптара чытаемага дыска;
* boot\_size – фіксаваны памер раздзела загрузкі;
* boot – аб“ект структуры FAT32\_BOOT, у каторым знаходзіцца счытаны раздзел загрузкі;
* fs\_info – аб“ект стуктуры FAT32\_FSInfo, у каторым знаходзіцца счытаны раздзел file system info;
* file\_sys – аб“ект стуктуры FAT32\_FS, у каторым знаходзіцца агульна важная інфармацыя па файлавай сістэме;
* bytes\_in\_sector – колькасць байтаў у сектары;
* sectors\_in\_сluster – колькасць сектараў у кластары;
* options – опцыі карыстальніка;
* total\_errors\_count – агульная колькасць знайдзеных памылак файлавай сістэмы;
* defected\_files – колькасць пашкоджаных файлаў;

**1.8 Файл fat\_struct.h**

У гэтым хэдэры знаходзіцца апісанне асноўных структур для працы з файлавай сістэмай FAT32. Структура FAT32\_BOOT апісвае раздзел загрузкі; FAT32\_FSInfo – раздзел file system info; DOS\_FILE – змяшчае спасылку на каталожны запіс, кластары бацькаўскага і прабацькоўскага каталога і шлях да файла; FAT32\_FS – змяшчае асноўную інфармацыю па файлавай сістэме; FAT\_ENTRY – запіс у файлавай табліцы; DIR\_ENT – каталожны запіс у раздзеле дадзеных.

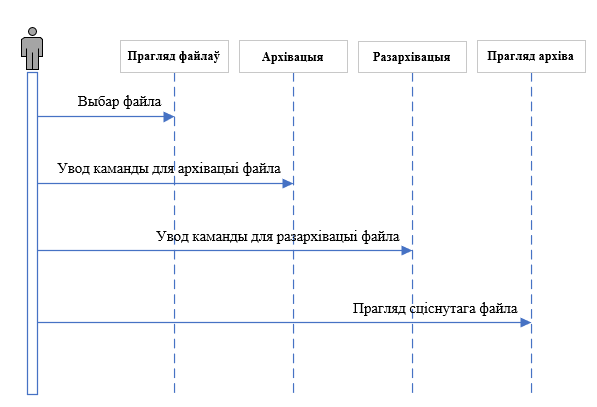
**3 ФУНКЦЫЯНАЛЬНАЕ ПРАЕКТАВАННЕ**

Для дакладнага разгляда кода і разумення праграмы ніжэй прадстаўлены алгарытм па кроках для асноўных функцый і дыяграма паслядоўнасцяў.

**3.1 Дыяграма паслядоўнасці**

Дыяграма паслядоўнасці – UML-дыяграма, каторая паказвае ўзаемадзеянне працэсаў у часовай паслядоўнасці пры распрацоўцы ці карыстанні праграмным забеспячэннем.

Дыяграмная паслядоўнасць распрацаванай прылады на малюнке 3.1.1



Малюнак 3.1.1 – UML-дыяграма паслядоўнасці.

Дадзеная дыяграма адлюстроўвае асноўныя функцыі праграмы і ўзаемадзеянне карыстальніка з праграмным функцыяналам.

**3.2 Функцыя int read\_boot()**

1. Ініцыялізаваць unsigned char\* buf і прысвоіць ёй значэнне (unsigned char\*)malloc(513 \* sizeof(unsigned char));
2. Вызваць функцыю void fs\_read(0, boot\_size, buf);
3. Ініцыялізаваць uint8\_t error і прысвоіць ёй значэнне 0;
4. Прысвоіць buf[boot\_size] значэнне 0;
5. Вызваць функцыю memcpy(&boot, buf, sizeof(struct FAT32\_BOOT));
6. Калі options & SHOW\_SECTORS, то перайсці на Крок 7, інакш Крок 8;
7. Вызваць функцыю pretty\_output();
8. Калі boot.sector\_size == 0 перайсці на Крок 9, інакш на Крок 11;
9. Вывесці чырвоным колерам у тэрмінал «Sector size is sero...»;
10. Павялічыць error на 1;
11. Калі boot.sectors\_in\_cluster == 0, то перайсці на Крок 12, інакш Крок 14;
12. Вывесці чырвоным колерам у тэрмінал "Cluster size is sero...\n";
13. Павялічыць error на 1;
14. Калі !boot.fats\_num && boot.fats\_num > 2, то перайсці на Крок 15, інакш Крок 17;
15. Вывесці чырвоным колерам у тэрмінал "Invalid count of fats\n";
16. Павялічыць error на 1;
17. Калі boot.sectors\_in\_table == 0, то перайсці на Крок 18, інакш Крок 20;
18. Вывесці чырвоным колерам у тэрмінал;
19. Павялічыць error на 1;
20. Калі total\_errors\_count = error, то перайсці на Крок 21, інакш Крок 23;
21. Вывесці чырвоным колерам у тэрмінал "Invalid boot sector\n";
22. Скончыць функцыю, вярнуўшы значэнне INVALID\_BOOT\_SECTOR;
23. Калі init\_file\_sys() == INVALID\_BOOT\_SECTOR, то перайсці на Крок 24, інакш Крок 26;
24. Вывесці чырвоным колерам у тэрмінал "Invalid boot sector\n";
25. Скончыць функцыю, вярнуўшы значэнне INVALID\_BOOT\_SECTOR;
26. Калі read\_fsinfo() == INVALID\_FSINFO, то перайсці на Крок 27, інакш Крок 29;
27. Вывесці чырвоным колерам у тэрмінал "Invalid fsinfo sector\n";
28. Скончыць функцыю, вярнуўшы значэнне INVALID\_FSINFO;
29. Калі options & SHOW\_INFO, то перайсці на Крок 30, інакш на Крок 33
30. Вызваць функцыю printf("\n\ndata start: %ld\n", file\_sys.data\_start);
31. Вызваць функцыю printf("data clusters count: %d - 0x%X\n", file\_sys.data\_clusters, file\_sys.data\_clusters);
32. Вызваць функцыю printf("fat\_start: %ld\n", file\_sys.fat\_start);
33. Скончыць функцыю, вярнуўшы нуль.

**3.3 Функцыя int read\_fsinfo()**

1. Ініцыялізаваць і прысвоіць 0;
2. Ініцыялізаваць unsigned char\* info\_buf і прысвоіць ёй значэнне (unsigned char\*)malloc(513 \* sizeof(unsigned char));
3. Вызваць функцыю void fs\_read(0, boot\_size, buf);
4. Прысвоіць buf[boot\_size] значэнне 0;
5. Вызваць функцыю memcpy(&fs\_info, info\_buf, sizeof(struct FAT32\_FSInfo));
6. Калі options & SHOW\_SECTORS, то перайсці на Крок 7, інакш на Крок 14;
7. Вызваць функцыю printf("\n\n\t ||\_\_\_\_FS INFO SECTOR\_\_\_\_||\n\n");
8. Вызваць функцыю printf("\tFIELD | |\tVALUE\n");
9. Вызваць функцыю printf("LEAD SIGNATURE | 0x%X\n", fs\_info.lead\_signature);
10. Вызваць функцыю printf("REVERSED SIGNATURE | 0x%X\n", fs\_info.rev\_signature);
11. Вызваць функцыю printf("FREE CLUSTERS COUNT | %d\n", fs\_info.free\_clusters);
12. Вызваць функцыю printf("NEXT CLUSTER NUMBER | %d\n", fs\_info.next\_cluster);
13. Вызваць функцыю printf("TRAIL SIGNATURE | 0x%X\n", fs\_info.trail\_signature);
14. Калі fs\_info.lead\_signature != 0x41615252, то перайсці на Крок 15, інакш на Крок 17;
15. Вызваць функцыю fprintf(stderr, RED "Invalid lead signature: 0x%X\n\n" RESET, fs\_info.lead\_signature);
16. Павялічыць error на 1;
17. Калі fs\_info.rev\_signature != 0x61417272, то перайсці на Крок 18, інакш на Крок 20;
18. Вызваць функцыю fprintf(stderr, RED "Invalid lead signature: 0x%X\n\n" RESET, fs\_info.lead\_signature);
19. Павялічыць error на 1;
20. Калі fs\_info.trail\_signature != 0xAA550000, то перайсці на Крок 21, інакш на Крок 23;
21. Вызваць функцыю fprintf(stderr, RED "Invalid trail signature: 0x%X\n\n" RESET, fs\_info.trail\_signature);
22. Павялічыць error на 1;
23. Павялічыць значэнне total\_errors\_count на error;
24. Калі error == 3, скончыць і вярнуць INVALID\_FS\_INFO, інакш скончыць і вярнуць 0;

**3.4 Функцыя int read\_fat()**

1. Ініцыіялізаваць int total\_clusters\_num і прысвоіць file\_sys.data\_clusters + 2;
2. Ініцыіялізаваць int eff\_size і прысвоіць (total\_clusters\_num \* file\_sys.entry\_size + 7) / 8ULL;
3. Ініцыіялізаваць void\* first і прысвоіць (unsigned int\*)malloc((eff\_size + 1)\* sizeof(void));
4. Ініцыіялізаваць void\* second і прысвоіць NULL;
5. Ініцыіялізаваць int first\_check і прысвоіць 0;
6. Ініцыіялізаваць int second\_check і прысвоіць 0;
7. Абвясціць аб’екты struct FAT\_ENTRY first\_media, second\_media;
8. Калі first\_check = check\_fat(first, &first\_media, file\_sys.fat\_start, eff\_size), то перайсці на Крок 9, інакш на Крок 10;
9. Вызваць функцыю fprintf(stderr, RED "first fat is invalid\n\n" RESET);
10. Калі file\_sys.fats\_num > 1, то перайсці на Крок 11, інакш на Крок 25;
11. Прысвоіць second значэнне (void\*)malloc((eff\_size + 1)\* sizeof(char));
12. Калі second\_check = check\_fat(second, &second\_media, file\_sys.fat\_start + file\_sys.fat\_size, eff\_size), то перайсці на Крок 13, інакш на Крок 14;
13. Вызваць функцыю fprintf(stderr, RED "second fat is invalid\n\n" RESET);
14. Калі !memcmp(first, second, eff\_size), то перайсці на Крок 15, інакш на Крок 16;
15. Вызваць функцыю printf(BLU "fats differs.\n\n" RESET);
16. Калі first\_check && !second\_check, то перайсці на Крок 17, інакш на Крок 19;
17. Вызваць функцыю free(first);
18. Прысвоіць first значэнне second;
19. Калі first\_check && second\_check, то перайсці на Крок 20, інакш на Крок 25;
20. Вызваць функцыю free(first);
21. Вызваць функцыю free(second);
22. Вызваць функцыю fprintf(stderr, RED "Invalid fats.\n\n" RESET);
23. Павялічыць total\_errors\_count на 1;
24. Скончыць функцыю, вярнуўшы INVALID\_FAT\_SECTOR;
25. Прысвоіць file\_sys.fat значэнне (unsigned char\*)malloc((eff\_size + 1) \* sizeof(unsigned char));
26. Вызваць функцыю memcpy(file\_sys.fat, (unsigned char\*)first, eff\_size);
27. Прысвоіць file\_sys.fat[eff\_size] значэнне 0;
28. Прысвоіць file\_sys.cluster\_owner значэнне malloc(total\_clusters\_num \* sizeof(struct DOS\_FILE\*));
29. Вызваць функцыю memset(file\_sys.cluster\_owner, 0, (total\_clusters\_num \* sizeof(struct DOS\_FILE\*)));
30. Вызваць функцыю free(second);
31. Скончыць функцыю, вярнуўшы 0;

**3.5 Функцыя void read\_disk(uint32\_t cluster, uint32\_t grandp, char\* file\_path)**

1. Ініцыялізаваць int sector\_num і прысвоіць 0;
2. Ініцыялізаваць int entry\_num і прысвоіць 0;
3. Ініцыялізаваць struct DIR\_ENT\* entry і прысвоіць (struct DIR\_ENT\*)malloc(1 \* sizeof(struct DIR\_ENT));
4. Ініцыялізаваць off\_t offset і прысвоіць base\_cluster(cluster);
5. Ініцыялізаваць struct DOS\_FILE\* file і прысвоіць (struct DOS\_FILE\*)malloc(sizeof(struct DOS\_FILE));
6. Прысвоіць file→dir\_ent значэнне (struct DIR\_ENT\*)malloc(sizeof(struct DIR\_ENT));
7. Прысвоіць file→parent\_cluster значэнне cluster;
8. Прысвоіць file→grandp\_cluster значэнне grandp;
9. Вызваць функцыю strcat(file\_path, "/");

**3.2 Метад void Encoder::Compression()**

1. Пачаць цыкл while(1) і прысвоіць \_hashHead значэнне 0;
2. Калі \_index + 2 < \_len, перайсці у функцыю InsertStringToHash(), інакш на Крок 3;
3. Прысвоіць \_prevLen значэнне \_matchLen;
4. Прысвоіць \_prevMatchStart значэнне \_matchStart;
5. Прысвоіць \_matchLen значэнне 2;
6. Калі \_hashHead != 0 і \_index - \_ hashHead <= \_len - 262 і \_prevLen < 16, перайсці на Крок 7, інакш Крок 10;
7. Прысвоіць \_matchLen значэнне функцыі LongestMatch();
8. Калі \_matchLen == 3 i \_index - \_matchStart > 4096, перайсці на Крок 9, інакш на Крок 10;
9. Прысвоіць \_matchLen = 2;
10. Калі \_prevLen >= 3 i \_matchLen <= \_prevLen, перайсці на Крок 11, інакш на Крок 29;
11. Ініціалізуем unsigned int distance і прысвоіць ёй значэнне \_index - \_prevMatchStart - 1;
12. Ініціалізуем unsigned int length і прысвоіць ёй значэнне \_prevLen - 3;
13. Прысвоіць \_symbolBuf[\_symbolNext] значэнне distance і павялічыць \_symbolNext на 1;
14. Прысвоіць \_symbolBuf[\_symbolNext] значэнне distance >> 8 і павялічыць \_symbolNext на 1;
15. Прысвоіць \_symbolBuf[\_symbolNext] значэнне length і павялічыць \_symbolNext на 1;
16. Паменшыць distance на 1;
17. Павялічыць  
     \_dynamicLiteralTree[\_length\_code[length] + 256 + 1].frequence на 1;
18. Павялічыць на 1 \_dynamicDistTree[d\_code(distance)].frequence;
19. Паменшыць \_prevLen на 2;
20. Павялічыць \_index на 1;
21. Калі \_index + 2 < \_len, перайсці на Крок 22, інакш на Крок 23;
22. Перайсці у функцыю InsertStringToHash();
23. Паменшыць \_prevLen на 1 і пачаць цыкл while(\_prevLen != 0);
24. Калі \_prevLen != 0, перайсці на Крок 20, інакш скончыць цыкл і перайсці на Крок 25;
25. Прысвоіць matchAvailable значэнне false;
26. Прысвоіць\_matchLen значэнне 2;
27. Павялічыць \_index на 1;
28. Калі \_index роўна \_len, перайсці на Крок 36 і скончыць цыкл на Кроку 1, інакш на Крок 1;
29. Калі matchaAvailable роўна true, перайсці на Крок 30, інакш на Крок 34;
30. Прысвоіць \_symbolBuf[\_symbolNext] значэнне 0 і павялічыць \_symbolNext на 1;
31. Прысвоіць \_symbolBuf[\_symbolNext] значэнне 0 і павялічыць \_symbolNext на 1;
32. Прысвоіць \_symbolBuf[\_symbolNext] значэнне \_uStr[\_index -1] і павялічыць \_symbolNext на 1;
33. Павялічыць \_index на 1;
34. Прысвоіць matchAvailable значэнне true;
35. Павялічыць \_index на 1 і перайсці на Крок 28;
36. Калі роўна, перайсці на Крок 37, інакш на Крок 41;
37. Прысвоіць \_symbolBuf[\_symbolNext] значэнне 0 і павялічыць \_symbolNext на 1;
38. Прысвоіць \_symbolBuf[\_symbolNext] значэнне 0 і павялічыць \_symbolNext на 1;
39. Прысвоіць \_symbolBuf[\_symbolNext] значэнне \_uStr[\_index -1] і павялічыць \_symbolNext на 1;
40. Павялічыць \_dynamicLiteralTree[\_uStr[\_index - 1]].frequence на 1;
41. Канец.

**3.3 Метад unsigned int Encoder::LongestMatch()**

1. Ініцыялізаваць unsigned int limit і прысвоіць ёй значэнне \_index - (\_len - 262), калі \_index > (\_len - 262), інакш 0;
2. Ініцыялізаваць unsigned char\* currStr і прысвоіць значэнне \_uStr + \_index;
3. Ініцыялізаваць unsigned int currMatchLen і прысвоіць ёй значэнне 0;
4. Ініцыялізаваць int bestPrevLen і прысвоіць ёй значэнне \_prevLen;
5. Ініцыялізаваць int matchLen і прысвоіць ёй значэнне 0;
6. Ініцыялізаваць int indexOfprev і прысвоіць ёйзначэнне 0;
7. Ініцыялізаваць int niceMatch і прысвоіць ёй значэнне 128;
8. Ініцыялізаваць unsigned char prevMatchLastChar і прысвоіць ёй значэнне currStr[bestPrevLen];
9. Ініцыялізаваць unsigned char prevMatchPreLastChar і прысвоіць ёй значэнне currStr[bestPrevLen - 1];
10. Ініцыялізаваць unsigned char\* maxMatxhStr і прысвоіць ёй значэнне\_uStr + \_index + 128;
11. Прысвоіць \_matchLen значэнне 0;
12. Прысвоіць matchLen значэнне 0;
13. Ініцыялізаваць unsigned char\* matchLen і прысвоіць ёй значэнне \_uStr + currMatchPos;
14. Прысвоіць \_hashHead значэнне currMatchPos;
15. Калі match[bestPrevLen] != prevMatchLastChar ці, match[bestPrevLen - 1] != prevMatchPreLastChar ці, match[0] != currStr[0] ці, match[1] != currStr[1], перайсці на Крок 16, інакш на Крок 19;
16. Прысвоіць indexOfprev значэнне \_prev[currMatchPos & \_winMask];
17. Прысвоіць currMatchPos значэнне indexOfPrev;
18. Перайсці на Крок32;
19. Ініцыялізаваць int і, Прысвоіць ёй значэнне 0;
20. Пачаць цыкл while(currStr != 0), калі currStr != 0 перайсці на Крок 21, інакш скончыць цыкл і перайсці на Крок 24;
21. Калі (match[i] != currStr[i] і currStr[i] != 0) ці (match >= maxMatchStr) ці matchLen == 258, выйсці з цыкла і перайсці на Крок 24, інакш перайсці на Крок 22;
22. Павялічыць i на 1;
23. Павялічыць matchLen на 1;
24. Калі matchLen > bestPrevLen, перайсці на Крок 25, інакш на Крок ;
25. Прысвоіць bestPrevLen значэнне matchLen;
26. Прысвоіць \_matchStart значэнне \_hashHead;
27. Калі bestPrevLen >= niceMatch, скончыць цыкл і перайсці на Крок 33, інакш перайсці на Крок 28;
28. Прысвоіць prevMatchLastChar значэнне currStr[bestPrevLen];
29. Прысвоіць prevMatchPreLastChar значэнне currStr[bestPrevLen - 1];
30. Прысвоіць indexOfprev значэнне \_prev[currMatchPos & \_winMask];
31. Прысвоіць currMatchPos значэнне indexOfprev;
32. Пачаць цыкл while(currMatchPos > limit), калі currMatchPos > limit перайсці на Крок 12, інакш скончыць цыкл і перайсці на  
    Крок 33;
33. Скончыць функцыю, каторая вяртае значэнне bestPrevLen.

**3.4 Метад void Encoder::BuildTree(TreeDesc& desc)**

1. Ініцыялізаваць huffmanTable\* dynTree і прысвоіць ёй значэнне desc.dynamicTree;
2. Ініцыялізаваць const huffmanTable\* staticTree і прысвоіць ёй значэнне desc.statDesc->staticTree;
3. Ініцыялізаваць int elems і прысвоіць ёй значэнне desc.statDesc->elems;
4. Ініцыялізаваць int maxCode і прысвоіць ёй значэнне -1;
5. Ініцыялізаваць int dadNode;
6. Прысвоіць \_heapLen значэнне 0;
7. Прысвойваць\_heapMaxLen значэнне 573;
8. Вызваць функцыю std::fill, каторая запаўняе масіў \_depth элементамі “Н”;
9. Пачаць цыкл for int i = 0; i < elems; i++;
10. Калі dynTree[i].frequence != 0, перайсці на Крок 11, інакш на Крок 13;
11. Павялічыць \_heapLen на 1 і прысвоіць \_heap[\_heapLen] і maxCode значэнне i;
12. Прысвоіць \_depth[i] значэнне 0;
13. Прысвоіць dynTree[i].len значэнне 0;
14. Калі i < elems, скончыць цыкл і перайсці на Крок 15, інакш на Крок 10;
15. Прысвоіць desc.maxCode значэнне maxCode;
16. Ініцыялізаваць int і і прысвоіць ёй значэнне 0;
17. Пачаць цыкл for i = \_heapLen / 2; i >= 1; i--
18. Вызваць функцыю DownHeap(dynTree, i);
19. Калі і >= 1, перайсці на Крок 17, інакш скончыць цыкл і перайсці на Крок 20;
20. Ініцыялізаваць int secondNode і прысвоіць ёй значэнне 0;
21. Ініцыялізаваць int firstNode і прысвоіць ёй значэнне 0;
22. Прысвоіць dadNode значэнне elems;
23. Прысвоіць firstNode значэнне \_heap[1];
24. Прысвоіць \_heap[1] значэнне \_heap[\_heapLen] і паменшыць \_heapLen на 1;
25. Вызваць функцыю DonwHeap(dynTree, 1);
26. Прысвоіць secondNode значэнне \_heap[1];
27. Паменшыць \_heapMaxLen на 1 і прысвоіць \_heap[\_heapMaxLen] значэнне firstNode;
28. Паменшыць \_heapMaxLen на 1 і прысвоіць \_heap[\_heapMaxLen] значэнне secondNode;
29. Прысвоіць dynTree[dadNode].frequence значэнне dynTree[firstNode].frequence + dynTree[secondNode].frequence;
30. Прысвоіць \_depth[dadNode] значэнне \_depth[firstNode] + 1, калі \_depth[firstNode] >= \_depth[secondNode], інакш \_depth[secondNode] + 1;
31. Прысвоіць dynTree[firstNode].dad і dynTree[secondNode].dad значэнне dadNode;
32. Прысвоіць \_heap[1] значэнне dadNode і павялічыць dadNode на 1;
33. Вызваць функцыі DownHeap(dynTree, 1);
34. Пачаць цыкл while(\_heap >= 2), калі \_heap >= 2 перайсці на Крок 23, інакш скончыць цыкл і перайсці на Крок 35;
35. Паменшыць \_heapMaxLen на 1 і прысвоіць \_heap[\_heapMaxLen] значэнне \_heap[1];
36. Канец функцыі.

**3.5 Метад void Encoder::DownHeap(huffmanTable\* dynTree, int i)**

1. Ініцыялізаваць int heapElem і прысвоіць ёй значэнне \_heap[i];
2. Ініцыялізаваць int j і прысвоіць ёй значэнне i << 1;
3. Пачаць цыкл while(j <= \_heapLen), калі j <= \_heapLen перайсці на Крок 4, інакш перайсці на Крок 10;
4. Калі j < \_heapLen і smaller(dynTree, \_heap[j + 1], \_heap[j], \_depth), перайсці на Крок 5, інакш на Крок 6;
5. Павялічыць j на 1;
6. Калі smaller(dynTree, heapElem, \_heap[j], \_depth), скончыць цыкл і перайсці на Крок 10, інакш на Крок 7;
7. Прысвоіць \_heap[i] значэнне \_heap[j];
8. Прысвоіць i значэнне j;
9. Прысвоіць j значэнне j << 1, перайсці на Крок 3;
10. Прысвоіць \_heap[i] значэнне heapElem;
11. Канец функцыі.

**3.6 Метад void Decoder::Decode()**

1. Ініцыялізаваць const code\* tablePtr;
2. Ініцыялізаваць unsigned int lengthMask і прысвоіць ёй значэнне (1 << \_lengthBits) - 1;
3. Ініцыялізаваць unsigned int distMask і прысвоіць ёй значэнне (1 << \_distBits) - 1;
4. Прысвоіць \_lastToRead значэнне \_encStr + (\_availableEnc - 5);
5. Прысвоіць \_end значэнне \_outStr + (\_availableOut - 257);
6. Калі , перайсці на Крок 7, інакш на Крок;
7. Прысвоіць \_bitBuffer значэнне \_bitBuffer + \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
8. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
9. Прысвоіць \_bitBuffer значэнне \_bitBuffer + \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
10. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
11. Прысвоіць tablePtr значэнне \_lengthCode + (\_bitBuffer & lengthMask);
12. Вызваць функцыю EncodeLength(tablePtr, lengthMask, distMask);
13. Пачаць цыкл while(\_encStr < \_lastToRead && \_outStr < \_end), калі (\_encStr < \_lastToRead && \_outStr < \_end) перайсці на Крок 6, інакш скончыць цыкл і перайсці на Крок 14;
14. Ініцыялізаваць unsigned int unusedBytes і прысвоіць значэнне \_bitsToEnc >> 3;
15. Прысвоіць \_encStr значэнне \_encStr - unusedBytes;
16. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc – unusedBytes << 3;
17. Прысвоіць \_bitBuffer значэнне \_bitBuffer & ((1 << \_bitsToEnc) -1);
18. Калі \_encStr < \_lastToRead, перайсці на Крок 19, інакш на Крок 20;
19. Прысвоіць \_availableEnc значэнне 5 + (\_lastToRead - \_encStr) і перайсці да Крока 21;
20. Прысвоіць \_availableEnc значэнне 5 - (\_encStr - \_lastToRead);
21. Калі \_outStr < \_end, перайсці на Крок 22, інакш на Крок 23;
22. Прысвоіць \_availableOut значэнне 257 + (\_end - \_outStr) і перайсці на Крок 24;
23. Прысвоіць \_availableOut значэнне 257 + (\_end - \_outStr);
24. Канец функцыі.

**3.7 Метад void Decoder::EncodeLength(const code\* table, unsigned int lenMask, unsigned int distance)**

1. Ініцыялізаваць length і прысвоіць ёй значэнне 0;
2. Ініцыялізаваць bits і прысвоіць ёй значэнне table->bits;
3. Ініцыялізаваць indicator і прысвоіць ёй значэнне table->indicator;
4. Прысвоіць \_bitBuffer значэнне \_bitBuffer >> bits;
5. Прысвоіць \_bitBuffer значэнне \_bitBuffer - bits;
6. Прысвоіць \_str значэнне \_outStr - \_lenOut;
7. Калі indicator роўна 0, перайсці на Крок 8, інакш на Крок 10;
8. Прысвоіць \*\_outStr значэнне table->value і павялічыць \_outStr на 1;
9. Павялічыць \_lenOut на 1 і перайсці на Крок 30;
10. Калі indicator & 16 != 0, перайсці на Крок 11, інакш на Крок 30;
11. Прысвоіць length значэнне table->value;
12. Прысвоіць indicator значэнне indicator & 15;
13. Калі indicator != 0, перайсці на Крок 14, інакш на Крок 20;
14. Калі \_bitsToEnc < indicator, перайсці на Крок 15, інакш на Крок 17;
15. Прысвоіць \_bitBuffer значэнне \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
16. Прысвоіць \_bitBuffer значэнне \_bitBuffer + 8;
17. Прысвоіць length значэнне length + \_bitBuffer & ((1 << indicator) - 1);
18. Прысвоіць \_bitBuffer значэнне \_bitBuffer >> indicator;
19. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc - indicator;
20. Калі \_bitsToEnc < 15, перайсці на Крок 21, інакш на Крок 25;
21. Прысвоіць \_bitBuffer значэнне \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
22. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
23. Прысвоіць \_bitBuffer значэнне \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
24. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
25. Прысвоіць table значэнне \_distCode + (\_bitBuffer & distMask);
26. Вызваць функцыю EncodeDist(table, length);
27. Калі, перайсці на Крок 28, інакш на Крок 30;
28. Прысвоіць table значэнне \_lengthCode + table->value + (\_bitBuffer & ((1 << indicator) - 1));
29. Вызваць функцыю EncodeLength(table, lenMask, distMask);
30. Канец функцыі.

**3.8 Метад void Decoder::EncoderDist(const code\* table, unsigned int length)**

1. Ініцыялізаваць unsigned int bits і прысвоіць значэнне table->bits;
2. Ініцыялізаваць unsigned int distance і прысвоіць значэнне 0;
3. Ініцыялізаваць unsigned int indicator і прысвоіць значэнне 0;
4. Прысвоіць \_bitBuffer значэнне \_bitBuffer >> bits;
5. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc - bits;
6. Прысвоіць indicator значэнне table->indicator;
7. Калі (indicator & 16) != 0, перайсці на Крок 8, інакш на Крок 24;
8. Прысвоіць distance значэнне table->value;
9. Прысвоіць indicator значэнне indicator & 15;
10. Калі \_bitsToEnc < indicator, перайсці на Крок 11, інакш на Крок 16;
11. Прысвоіць \_bitBuffer значэнне \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
12. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
13. Калі \_bitsToEnc < indicator, перайсці на Крок 14, інакш на Крок 16;
14. Прысвоіць \_bitBuffer значэнне \*\_encStr << \_bitsToEnc і павялічыць \_encStr на 1;
15. Прысвоіць \_bitsToEnc значэнне \_bitsToEnc + 8;
16. Прысвоіць distance значэнне distance + \_bitBuffer & ((1 << indicator) - 1);
17. Прысвоіць \_bitBuffer значэнне \_bitBuffer >> indicator;
18. Прысвоіць \_bitToEnc значэнне \_bitsToEnc - indicator;
19. Ініцыялізаваць unsigned char\* readFrom і прысвоіць ёй значэнне \_outStr - distance;
20. Пачаць цыкл for int i = 0; i < length; i++;
21. Прысвоіць \*\_outStr значэнне \*readFrom і павялічыць \*\_outStr і \*readFrom на 1;
22. Павялічыць \_lenOut на 1;
23. Калі i < length, перайсці на Крок 21, інакш на Крок 27;
24. Калі (indicator & 64) = 0, перайсці на Крок 25, інакш на Крок 27;
25. Прысвоіць table значэнне \_distCode + table->value + (\_bitBuffer & ((1 << indicator) - 1));
26. Вызваць функцыю EncodeDist(table, length);
27. Канец функцыі.

**3.9 Метад void Encoder::InsertStringToHash()**

1. Прысвоіць \_insertHash значэнне (((\_insertHash << \_hashShift) ^ (\_uStr[\_index + 2])) & \_hashMask);
2. Прысвоіць \_hashHead i \_prev[\_index & \_winMask] значэнне \_head[\_insertHash];
3. Прысвоіць \_head[\_insertHash] значэнне \_index;
4. Канец функцыі.

**4 ІНСТРУКЦЫЯ КАРЫСТАЛЬНІКА**

Як ужо было напісана раней, праграма архіватара працуе ў два бакі – архівацыя і разархівацыя. Для таго каб заархіваваць ці разархіваваць файл, трэба памясціць яго ў папку з EXE.

Пасля гэтага, трэба выбраць, якую задачу жадаеце выканаць.

Для архівацыі:

1. У камандны радок напісаць імя EXE-файла архіватара;
2. Націснуць прабел і прапісаць імя файла, каторы хочаце скампрэсаваць;
3. Націснуць прабел і ўвесці імя файла, у каторы хочаце запісаць вынік архівацыі.

Для разархівацыі:

1. У камандны радок напісаць імя EXE-файла архіватара;
2. Націснуць прабел і напісаць каманду “-d”.
3. Націснуць прабел і прапісаць імя файла, каторы хочаце дэкампрэсаваць;
4. Націснуць прабел і ўвесці імя файла, у каторы хочаце запісаць вынік разархівацыі.

Візуальны прыклад прыведзены ў скрыншотах у Прыдатку Б.

**ЗАКЛЮЧЭННЕ**

У дадзеным курсавым праекце была рэалізавана праграма-архіватар на аснове вывучанага матэрыяла і ведаў з дысцыпліны “Праграмаванне на мовах высокага ўзроўня”, аформлены блок-схема алгарытмаў і дыяграма класаў.

Вялізная тэма архівацыі закранула алгарытмы сартыроўкі, паняцце хэшырвання і хэш-табліцы, праектаванне праграмнага забеспячэння і некаторыя раздзелы матэматыкі. Складанасць рэалізацыі праграмы і пошука дакладнай інфармацыі па алгарытму і спосабу запісу ў файл, прымусіла палепшыць веды англійскай мовы, сістэм злічэння і абстрактнага мыслення.

Асноўнымі плюсамі напісанай праграмы з’яўляюцца: хуткасць, аптымізаванасць і сумяшчальнасць з афіцыйным архіватарам ад ZLIB[7](файл, каторы сціснуты маім архіватарам, можа разархіваваць ZLIB).

У далейшым гэты праект можна ўдасканаліць у наступных аспектах:

1. Дадаць магчымасць сціскасць і разархіваваць файлы з памерам больш за 16 кілабайт.
2. Удасканаліць алгарытмы кампрэсіі.
3. Напісаць графічны інтэрфейс для кіравання ўсімі заархіваванымі файламі з дадатковымі функцыямі.

**СПІС ВЫКАРЫСТАВАНЫХ КРЫНІЦ**

[1] Understanding zlib [Электронны рэсурс]. – Рэжым доступа: <https://www.euccas.me/zlib/>. – [Дата доступа] – 25.09.2022.

[2] ZLIB Compressed Data Format Specification version 3.3 [Электронны рэсурс]. – Рэжым доступа: <https://www.rfc-editor.org/rfc/rfc1950>.

[3] DEFLATE Compressed Data Format Specification version 1.3 [Электронны рэсурс]. – Рэжым доступа: https://www.rfc-editor.org/rfc/rfc1951.

[4] An Explanation of the Deflate Algorithm [Электронны рэсурс]. – Рэжым доступа: <https://www.zlib.net/feldspar.html>. – [Дата доступа] – 10.10.2022.

[5] Ватолін, Д. Метады сціску дадзеных. Будова архіватараў, сціск відарысаў і відэа : вучэбна-справачны дапаможнік / Д. Ватолін [і інш.]. – M.: ДИАЛОГ-МИФИ, 2003. – 384с. – №1. – C. 77 – 86.

[6] Data Compression - Lecture 11 - DEFLATE (gzip) [Электронны рэсурс]. – Рэжым доступа : [https://www.youtube.com/watch?v=oi2lMBBjQ8s](https://www.youtube.com/watch?v=oi2lMBBjQ8s&t=5009s). – [Дата доступа] – 15.10.2022.

[7] zlib Home Site [Электронны рэсурс]. – Рэжым доступа: <https://www.zlib.net/>. – [Дата доступа] – 20.10.2022.

**ПРЫДАТАК А**

*(абавязковы)*

Лісцінг кода.

\\ файд “Source.cpp”

#include <iostream>

#include <fstream>

#include <string>

#include "Container.h"

#include "ILogger.h"

#include "Exceptions.h"

#include "ICoder.h"

#include "Encoder.h"

#include "Decoder.h"

#define nameof(name) #name

#define \_EXCEPTION\_OUTPUT(ex)\

(\

(std::cout << "Error message: " << ex.what() << std::endl), \

(std::cout << "Error code: " << ex.GetErrorCode() << std::endl) \

)

// функцыя праверкі назвы файлы

void CheckFileName(char\* fileName)

{

char forbidden[9] = { '<', '>', ':', '\"', '/', '\\', '|', '?', '\*' };

int length = strlen(fileName);

int i = 0;

while (i != length)

{

for (int j = 0; j < 9; j++)

{

if (fileName[i] == forbidden[j])

{

throw FileNameException(fileName);

}

}

i++;

}

}

// функцыя праверкі файлаў на адкрыццё і назву

short CheckFiles(Logger& logger, File& logFile, File& input, File& output, char\*\* argv)

{

try

{

logFile.file.open(logFile.fileName, std::ios::app | std::ios::binary);

if (!logFile.file.is\_open())

{

throw FileOpenErrorException(logFile.fileName);

}

}

catch (FileOpenErrorException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

return -1;

}

logFile.file.close();

try

{

CheckFileName(argv[1]);

input.file.open(argv[1], std::ios::binary | std::ios::in);

if (!input.file.is\_open())

{

throw FileOpenErrorException(argv[1]);

}

}

catch (FileNameException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

catch (FileOpenErrorException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

input.file.close();

input.fileName = argv[1];

input.fileDesc = 3;

std::string send = argv[1];

logger.LogInfo(send + " is opened", logFile);

send = "";

try

{

CheckFileName(argv[2]);

output.file.open(argv[2], std::ios::binary | std::ios::out);

if (!output.file.is\_open())

{

throw FileOpenErrorException(argv[2]);

}

}

catch (FileNameException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

catch (FileOpenErrorException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

output.file.close();

output.fileName = argv[2];

output.fileDesc = 2;

send = argv[2];

logger.LogInfo(send + " is opened", logFile);

return 0;

}

int main(int argc, char\*\* argv)

{

// вывад файла з тлумачэннем

if (argc == 1 || argc == 2)

{

std::ifstream explanation("README.txt", std::ios::binary);

std::string str;

std::cout << std::endl;

if (explanation.is\_open())

{

while (explanation)

{

std::getline(explanation, str);

std::cout << str << std::endl;

}

}

explanation.close();

return 0;

}

// кампрэсія

if (argc == 3 && strcmp(argv[1], "-d") != 0)

{

std::string logName = "D:\\Course project\\Course\\LOGGER.txt";

File logFile(logName, 1);

File input;

File output;

Logger;

if (CheckFiles(logger, logFile, input, output, argv))

{

return -1;

};

FileContainer container;

container.SetTop(container.PushFile(input, nullptr, nullptr));

container.PushFile(output, container.GetTop(), nullptr);

container.PushFile(logFile, container.GetTop(), nullptr);

Encoder enc(logger, container);

try

{

enc.Compression();

}

catch (FileSizeOutOfRangeException ex)

{

rewind(stdin);

\_EXCEPTION\_OUTPUT(ex);

logger.LogWarning(ex.what(), logFile);

return -1;

}

logger.LogInfo(" successful compression.", logFile);

return 0;

}

// дэкампрэсія

if (argc == 4 && strcmp(argv[1], "-d") == 0)

{

std::string logName = "D:\\Course project\\Course\\LOGGER.txt";

File logFile(logName, 1);

File input;

File output;

Logger;

argv += 1;

if (CheckFiles(logger, logFile, input, output, argv))

{

return -1;

}

FileContainer container;

container.SetTop(container.PushFile(input, nullptr, nullptr));

container.PushFile(output, container.GetTop(), nullptr);

container.PushFile(logFile, container.GetTop(), nullptr);

Decoder dec(logger, container);

try

{

dec.Decompression();

}

catch (InvalidMagicException ex)

{

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

catch (InvalidHeaderException ex)

{

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

catch (InvalidAdlerException ex)

{

\_EXCEPTION\_OUTPUT(ex);

logger.LogError(ex.what(), logFile);

return -1;

}

logger.LogInfo(" successful compression.", logFile);

}

return 0;

}

\\ файл “Container.h”

#pragma once

#include <iostream>

#include <fstream>

struct File

{

std::string fileName; // імя файл

unsigned short fileDesc; // файлавы дэскрыптар

std::fstream file;

// канструктары

File() = default;

File(std::string& name, unsigned short desc)

{

fileName = name;

fileDesc = desc;

}

File(const File& Data)

{

fileName = Data.fileName;

fileDesc = Data.fileDesc;

}

void operator=(const File&);

};

struct FileNode

{

File data;

FileNode\* left; // паказальнік на левы элемент

FileNode\* right; // паказальнік на правы элемент

FileNode\* dad; // паказальнік на бацькоўскі вузел

// канструктары

FileNode(const File&);

FileNode() = delete;

};

class FileContainer

{

private:

FileNode\* \_top;

public:

FileContainer() : \_top(nullptr){} // канструктар

~FileContainer() = default; // дэструктар

void SetTop(FileNode\* top)

{

\_top = top;

}

FileNode\* GetTop()

{

return \_top;

}

FileNode\* PushFile(File, FileNode\*, FileNode\*);

class Iterator

{

private:

FileNode\* \_initialNode; // бягучы вузел

FileNode\* \_topOfCon; // корань кантэйнера

public:

// канструктары

Iterator() : \_initialNode(nullptr), \_topOfCon(nullptr)

{

}

Iterator(FileNode\* inNode, FileNode\* top) : \_initialNode(inNode), \_topOfCon(top)

{

}

FileNode\* GetInNode()

{

return \_initialNode;

}

File& operator++(int b);

File& operator--(int b);

File& operator\*()

{

return \_initialNode->data;

}

bool operator==(const Iterator& b);

bool operator!=(const Iterator& b);

};

Iterator Begin();

Iterator End();

};

\\ файл “Container.cpp”

#include <iostream>

#include "Container.h"

void File::operator=(const File& b)

{

fileName = b.fileName;

fileDesc = b.fileDesc;

}

FileNode::FileNode(const File& Data) : data(Data), left(nullptr), right(nullptr), dad(nullptr)

{

}

// функцыя дабаўлення файла ў кантэйнер

FileNode\* FileContainer::PushFile(File data, FileNode\* node, FileNode\* dad)

{

if (node == nullptr)

{

FileNode\* node = new FileNode(data);

return node;

}

if (node->data.fileDesc > data.fileDesc)

{

node->left = PushFile(data, node->left, node);

}

if (node->data.fileDesc < data.fileDesc)

{

node->right = PushFile(data, node->right, node);

}

return node;

}

// устаноўка ітэратара на пачатак кантэйнера

FileContainer::Iterator FileContainer::Begin()

{

if (\_top->right != nullptr)

{

Iterator iter(\_top->right, \_top);

return iter;

}

Iterator iterT(\_top, \_top);

return iterT;

}

// устаноўка ітэратара на канец кантэйнера

FileContainer::Iterator FileContainer::End()

{

Iterator iter(\_top->left->left, \_top);

return iter;

}

\\ файл “ContainerIterator.cpp”

#include <iostream>

#include "Container.h"

// функцыя рушэння ітэратара ў канец кантэйнера

File& FileContainer::Iterator::operator++(int b)

{

if (\_initialNode->data.fileDesc > \_topOfCon->data.fileDesc)

{

\_initialNode = \_topOfCon;

return \_initialNode->data;

}

if (\_initialNode->data.fileDesc == \_topOfCon->data.fileDesc)

{

\_initialNode = \_initialNode->left;

return \_initialNode->data;

}

if (\_initialNode->left == nullptr)

{

return \_initialNode->data;

}

\_initialNode = \_initialNode->left;

return \_initialNode->data;

}

// функцыя рушэння ітэратара ў пачатак кантэйнера

File& FileContainer::Iterator::operator--(int b)

{

if (\_initialNode->data.fileDesc < \_topOfCon->data.fileDesc)

{

if (\_initialNode->left == nullptr)

{

\_initialNode = \_initialNode->dad;

return \_initialNode->data;

}

\_initialNode = \_topOfCon;

return \_initialNode->data;

}

if (\_initialNode->data.fileDesc == \_topOfCon->data.fileDesc)

{

\_initialNode = \_topOfCon;

return \_initialNode->data;

}

\_initialNode = \_topOfCon->right;

return \_initialNode->data;

}

// функцыі параўнання дзвюх элементаў

bool FileContainer::Iterator::operator!=(const Iterator& b)

{

if (\_initialNode->data.fileDesc != b.\_initialNode->data.fileDesc)

{

return true;

}

return false;

}

bool FileContainer::Iterator::operator==(const Iterator& b)

{

if (\_initialNode->data.fileDesc == b.\_initialNode->data.fileDesc)

{

return true;

}

return false;

}

\\ файл “ILogger.h”

#pragma once

#include <iostream>

#include <chrono>

#include "Container.h"

class ILoggerManager

{

private:

std::string \_logMessage;

public:

virtual void LogInfo(std::string msg, File& data) = 0;

virtual void LogWarning(std::string msg, File& data) = 0;

virtual void LogError(std::string msg, File& data) = 0;

// запіс у файл

void WriteToFile(File& data)

{

data.file.open(data.fileName, std::ios::binary | std::ios::app);

data.file << \_logMessage;

data.file << std::endl;

data.file.close();

}

// удасканальванне паведамлення

void SetLogMessage(std::string msg, File& data)

{

auto time = std::chrono::system\_clock::now();

time\_t logTime = std::chrono::system\_clock::to\_time\_t(time);

\_logMessage = std::ctime(&logTime) + msg;

WriteToFile(data);

}

};

class Logger : public ILoggerManager

{

public:

// метады адсылкі паведамленняў

void LogInfo(std::string msg, File& data)

{

ILoggerManager::SetLogMessage(" - Info: " + msg, data);

}

void LogWarning(std::string msg, File& data)

{

ILoggerManager::SetLogMessage(" - Warning: " + msg, data);

}

void LogError(std::string msg, File& data)

{

ILoggerManager::SetLogMessage(" - Error: " + msg, data);

}

};

\\ файл “Exceptions.h”

#pragma once

#include <iostream>

#include <cstdlib>

class Exception : public std::exception

{

private:

std::string \_errorCode;

public:

Exception(std::string msg) : exception(msg.c\_str())

{

}

void SetErrorCode(const std::string& errorCode)

{

\_errorCode = errorCode;

}

std::string& GetErrorCode()

{

return \_errorCode;

}

};

// выключэнне на адкрыццё файла

class FileOpenErrorException : public Exception

{

private:

std::string \_errorCode = "F101X";

public:

FileOpenErrorException(std::string msg) : Exception("File " + msg + " cannot be opened.")

{

Exception::SetErrorCode(\_errorCode);

}

};

// выключэнне на няправільную назву файла

class FileNameException : public Exception

{

private:

std::string \_errorCode = "F102X";

public:

FileNameException(std::string msg) : Exception(msg + " is invalid for file name.")

{

Exception::SetErrorCode(\_errorCode);

}

};

// выключэнне на адстунасць “магічнай” лічбы

class InvalidMagicException : public Exception

{

private:

std::string \_errorCode = "FMAGICX";

public:

InvalidMagicException(std::string msg) : Exception(msg + " - invalid magic number.")

{

Exception::SetErrorCode(\_errorCode);

}

};

class InvalidHeaderException : public Exception

{

private:

std::string \_errorCode = "FHEADERX";

public:

InvalidHeaderException(std::string msg) : Exception(msg + " - invalid header.")

{

Exception::SetErrorCode(\_errorCode);

}

};

class InvalidAdlerException : public Exception

{

private:

std::string \_errorCode = "FADLERX";

public:

InvalidAdlerException(std::string msg) : Exception(msg + " - invalid adler.")

{

Exception::SetErrorCode(\_errorCode);

}

};

class FileSizeOutOfRangeException : public Exception

{

private:

std::string \_errorCode = "FSIZEX";

public:

FileSizeOutOfRangeException(std::string msg) : Exception(msg + " - file size is out of range(1 to 16384).")

{

Exception::SetErrorCode(\_errorCode);

}

};

\\ файл “ICoder.h”

#pragma once

#include <iostream>

#include "ILogger.h"

#include "Container.h"

#define STATIC\_TREES 1 // код статычных табліц

#define DYN\_TREES 2 // код дынамічных табліц

#define LENGTH\_CODES 29 // колькасць длін

#define LITERALS 256 // колькасць сімвалаў

#define L\_CODES (LITERALS + 1 + LENGTH\_CODES) // сімвалы + дліны

#define D\_CODES 30 // колькасць дыстанцый

#define BL\_CODES 19 // колькасць длін кодаў.

#define HEAP\_SIZE (2 \* L\_CODES + 1) // памер кучы

#define MAX\_BL\_BITS 7 // максімальны памер дліны кода "длін кодаў"

#define MAX\_BITS 15 // максімальная колькасць выкарыставанных бітоў

#define MAX\_MATCH 258 // максімальная дліна супадзення

#define MIN\_MATCH 3 // мінімальная дліна супадзення

#define MAX\_DIST(winSize) (winSize - 262) // максімальная дыстанцыя

#define d\_code(dist) \

((dist) < 256 ? \_dist\_code[dist] : \_dist\_code[256 + ((dist) >> 7)]) \

#define END\_BLOCK 256 // сімвал канца блока

// Adler defines

#define BASE 65521

#define NMAX 5552

// паступовае складанне наступных 16 элементаў радка

#define DO1(buf,i) {adler += (buf)[i]; sum += adler;}

#define DO2(buf,i) DO1(buf,i); DO1(buf,i+1);

#define DO4(buf,i) DO2(buf,i); DO2(buf,i+2);

#define DO8(buf,i) DO4(buf,i); DO4(buf,i+4);

#define DO16(buf) DO8(buf,0); DO8(buf,8);

// вызначаем меншы элемент па колькасць сустрэч у табліцы

#define smaller(Tree, n, m, depth) \

(Tree[n].frequence < Tree[m].frequence || \

(Tree[n].frequence == Tree[m].frequence && depth[n] <= depth[m]))

class ICoder

{

public:

virtual unsigned long Adler32(unsigned char\* str, unsigned long adler, unsigned int len);

virtual unsigned int ReadFromFile(FileContainer& container, unsigned char\* str);

virtual void WriteToFile(FileContainer& container, std::string str);

};

\\ файл “ICoder.cpp”

#include <iostream>

#include "ICoder.h"

#include "Container.h"

#include "Exceptions.h"

// функцыя падлічвання чэк-сумы

unsigned long ICoder::Adler32(unsigned char\* str, unsigned long adler, unsigned int len)

{

unsigned int length = len;

str[len] = 0;

unsigned long sum;

unsigned int n;

sum = (adler >> 16) & 0xffff;

adler &= 0xffff;

if (length == 1)

{

adler += str[0];

if (adler == BASE)

{

adler -= BASE;

}

sum += adler;

if (sum >= BASE)

{

sum -= BASE;

}

return adler | (sum << 16);

}

while (length >= NMAX)

{

length -= NMAX;

n = NMAX / 16;

do

{

DO16(str);

str += 16;

} while (--n);

adler %= BASE;

sum %= BASE;

}

if (length != 0)

{

while (length >= 16)

{

length -= 16;

DO16(str);

str += 16;

}

while (length--)

{

adler += \*str++;

sum += adler;

}

adler %= BASE;

sum %= BASE;

}

return adler | (sum << 16);

}

// функцыя счытвання з файла

unsigned int ICoder::ReadFromFile(FileContainer& container, unsigned char\* str)

{

FileContainer::Iterator iter;

// пошук патрэбнага файла

for (iter = container.Begin(); iter != container.End(); iter++)

{

if (iter.GetInNode() == container.GetTop())

{

break;

}

}

FileNode\* node = iter.GetInNode();

File readFile = node->data;

readFile.file.open(readFile.fileName, std::ios::binary | std::ios::in);

readFile.file.seekg(0, std::ios::end);

unsigned int fileSize = readFile.file.tellg();

if (fileSize <= 1 || fileSize > 16384)

{

throw FileSizeOutOfRangeException(readFile.fileName);

}

readFile.file.clear();

readFile.file.seekg(0, readFile.file.beg);

readFile.file.read((char\*)str, fileSize);

readFile.file.close();

rewind(stdin);

return fileSize;

}

// функцыя запісу ў файл

void ICoder::WriteToFile(FileContainer& container, std::string str)

{

FileContainer::Iterator iter;

// пошук патрэбнага файла

for (iter = container.Begin(); iter != container.End(); iter++)

{

if (iter.GetInNode() == container.GetTop()->left)

{

break;

}

}

FileNode\* node = iter.GetInNode();

File writeFile = node->data;

writeFile.file.open(writeFile.fileName, std::ios::binary | std::ios::out);

writeFile.file << str;

writeFile.file.close();

rewind(stdin);

}

\\ файл “Encoder.h”

#pragma once

#include "ICoder.h"

struct huffmanTable

{

unsigned short frequence; // колькасць паўтораў пэўнага сімвала

unsigned short code; // код элемента

unsigned short dad; // бацькоўскі вузел

unsigned short len; // дліна кода

};

struct StaticTreeDesc

{

const huffmanTable\* staticTree; // паказальнік на статычную табліцу (дрэва)

const int\* extraBits; // дадатковая біты

int extraBase; // дадатковы пачатак для табліцы длін

int elems; // колькасць элементаў у табліцы (дрэве)

int maxBitLen; // максімальная дліна кода

};

struct TreeDesc

{

huffmanTable\* dynamicTree; // паказальнік дынамічнага дрэва

int maxCode; // максімальнае значэнне кода

const StaticTreeDesc\* statDesc; // паказальнік на апісанне для стат дрэва

};

// табліца кадоў длін

const unsigned char \_length\_code[MAX\_MATCH - MIN\_MATCH + 1] = {

0, 1, 2, 3, 4, 5, 6, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 12, 12,

13, 13, 13, 13, 14, 14, 14, 14, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16,

17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 19, 19, 19, 19,

19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,

21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22, 22, 22, 22,

22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,

23, 23, 23, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,

24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,

25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,

25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26,

26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26,

26, 26, 26, 26, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27,

27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 28

};

// табліца кадоў дыстанцый

const unsigned char \_dist\_code[512] = {

0, 1, 2, 3, 4, 4, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8,

8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10,

10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,

11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,

12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13,

13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,

13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,

14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,

14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,

14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 15, 15,

15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,

15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,

15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 0, 0, 16, 17,

18, 18, 19, 19, 20, 20, 20, 20, 21, 21, 21, 21, 22, 22, 22, 22, 22, 22, 22, 22,

23, 23, 23, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,

24, 24, 24, 24, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,

26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26,

26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27, 27, 27, 27, 27, 27, 27,

27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27,

27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,

28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,

28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,

28, 28, 28, 28, 28, 28, 28, 28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29,

29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29,

29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29,

29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29

};

// стандартныя (без дад. бітоў) значэнні длін

const int base\_length[LENGTH\_CODES] =

{

0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 24, 28, 32, 40, 48, 56,

64, 80, 96, 112, 128, 160, 192, 224, 0

};

// стандартныя (без дад. бітоў) значэнні дыстанцый

const int base\_dist[D\_CODES] =

{

0, 1, 2, 3, 4, 6, 8, 12, 16, 24,

32, 48, 64, 96, 128, 192, 256, 384, 512, 768,

1024, 1536, 2048, 3072, 4096, 6144, 8192, 12288, 16384, 24576

};

// статычныя табліцы для дыстанцый і сімвалаў/длін

const huffmanTable static\_dtree[D\_CODES] = {

{{ 0}, { 0}, { 5}, { 5}}, {{16}, {16}, { 5}, { 5}}, {{ 8}, { 8}, { 5}, { 5}}, {{24}, {24}, { 5}, { 5}}, {{ 4}, { 4}, { 5}, { 5}},

{{20}, {20}, { 5}, { 5}}, {{12}, {12}, { 5}, { 5}}, {{28}, {28}, { 5}, { 5}}, {{ 2}, { 2}, { 5}, { 5}}, {{18}, {18}, { 5}, { 5}},

{{10}, {10}, { 5}, { 5}}, {{26}, {26}, { 5}, { 5}}, {{ 6}, { 6}, { 5}, { 5}}, {{22}, {22}, { 5}, { 5}}, {{14}, {14}, { 5}, { 5}},

{{30}, {30}, { 5}, { 5}}, {{ 1}, { 1}, { 5}, { 5}}, {{17}, {17}, { 5}, { 5}}, {{ 9}, { 9}, { 5}, { 5}}, {{25}, {25}, { 5}, { 5}},

{{ 5}, { 5}, { 5}, { 5}}, {{21}, {21}, { 5}, { 5}}, {{13}, {13}, { 5}, { 5}}, {{29}, {29}, { 5}, { 5}}, {{ 3}, { 3}, { 5}, { 5}},

{{19}, {19}, { 5}, { 5}}, {{11}, {11}, { 5}, { 5}}, {{27}, {27}, { 5}, { 5}}, {{ 7}, { 7}, { 5}, { 5}}, {{23}, {23}, { 5}, { 5}}

};

const huffmanTable static\_ltree[L\_CODES + 2] = {

{{ 12}, { 12}, { 8}, { 8}}, {{140}, {140}, { 8}, { 8}}, {{ 76}, { 76}, { 8}, { 8}}, {{204}, {204}, { 8}, { 8}}, {{ 44}, { 44}, { 8}, { 8}},

{{172}, {172}, { 8}, { 8}}, {{108}, {108}, { 8}, { 8}}, {{236}, {236}, { 8}, { 8}}, {{ 28}, { 28}, { 8}, { 8}}, {{156}, {156}, { 8}, { 8}},

{{ 92}, { 92}, { 8}, { 8}}, {{220}, {220}, { 8}, { 8}}, {{ 60}, { 60}, { 8}, { 8}}, {{188}, {188}, { 8}, { 8}}, {{124}, {124}, { 8}, { 8}},

{{252}, {252}, { 8}, { 8}}, {{ 2}, { 2}, { 8}, { 8}}, {{130}, {130}, { 8}, { 8}}, {{ 66}, { 66}, { 8}, { 8}}, {{194}, {194}, { 8}, { 8}},

{{ 34}, { 34}, { 8}, { 8}}, {{162}, {162}, { 8}, { 8}}, {{ 98}, { 98}, { 8}, { 8}}, {{226}, {226}, { 8}, { 8}}, {{ 18}, { 18}, { 8}, { 8}},

{{146}, {146}, { 8}, { 8}}, {{ 82}, { 82}, { 8}, { 8}}, {{210}, {210}, { 8}, { 8}}, {{ 50}, { 50}, { 8}, { 8}}, {{178}, {178}, { 8}, { 8}},

{{114}, {114}, { 8}, { 8}}, {{242}, {242}, { 8}, { 8}}, {{ 10}, { 10}, { 8}, { 8}}, {{138}, {138}, { 8}, { 8}}, {{ 74}, { 74}, { 8}, { 8}},

{{202}, {202}, { 8}, { 8}}, {{ 42}, { 42}, { 8}, { 8}}, {{170}, {170}, { 8}, { 8}}, {{106}, {106}, { 8}, { 8}}, {{234}, {234}, { 8}, { 8}},

{{ 26}, { 26}, { 8}, { 8}}, {{154}, {154}, { 8}, { 8}}, {{ 90}, { 90}, { 8}, { 8}}, {{218}, {218}, { 8}, { 8}}, {{ 58}, { 58}, { 8}, { 8}},

{{186}, {186}, { 8}, { 8}}, {{122}, {122}, { 8}, { 8}}, {{250}, {250}, { 8}, { 8}}, {{ 6}, { 6}, { 8}, { 8}}, {{134}, {134}, { 8}, { 8}},

{{ 70}, {70}, { 8}, { 8}}, {{198}, {198}, { 8}, { 8}}, {{ 38}, { 38}, { 8}, { 8}}, {{166}, {166}, { 8}, { 8}}, {{102}, {102}, { 8}, { 8}},

{{230}, {230}, { 8}, { 8}}, {{ 22}, { 22}, { 8}, { 8}}, {{150}, {150}, { 8}, { 8}}, {{ 86}, { 86}, { 8}, { 8}}, {{214}, {214}, { 8}, { 8}},

{{ 54}, { 54}, { 8}, { 8}}, {{182}, {182}, { 8}, { 8}}, {{118}, {118}, { 8}, { 8}}, {{246}, {246}, { 8}, { 8}}, {{ 14}, { 14}, { 8}, { 8}},

{{142}, {142}, { 8}, { 8}}, {{ 78}, { 78}, { 8}, { 8}}, {{206}, {206}, { 8}, { 8}}, {{ 46}, { 46}, { 8}, { 8}}, {{174}, {174}, { 8}, { 8}},

{{110}, {110}, { 8}, { 8}}, {{238}, {238}, { 8}, { 8}}, {{ 30}, { 30}, { 8}, { 8}}, {{158}, {158}, { 8}, { 8}}, {{ 94}, { 94}, { 8}, { 8}},

{{222}, {222}, { 8}, { 8}}, {{ 62}, { 62}, { 8}, { 8}}, {{190}, {190}, { 8}, { 8}}, {{126}, {126}, { 8}, { 8}}, {{254}, {254}, { 8}, { 8}},

{{ 1}, { 1}, { 8}, { 8}}, {{129}, {129}, { 8}, { 8}}, {{ 65}, { 65}, { 8}, { 8}}, {{193}, {193}, { 8}, { 8}}, {{ 33}, { 33}, { 8}, { 8}},

{{161}, {161}, { 8}, { 8}}, {{ 97}, { 97}, { 8}, { 8}}, {{225}, {225}, { 8}, { 8}}, {{ 17}, { 17}, { 8}, { 8}}, {{145}, {145}, { 8}, { 8}},

{{ 81}, {81 }, { 8}, { 8}}, {{209}, {209}, { 8}, { 8}}, {{ 49}, { 49}, { 8}, { 8}}, {{177}, {177}, { 8}, { 8}}, {{113}, {113}, { 8}, { 8}},

{{241}, {241}, { 8}, { 8}}, {{ 9}, { 9}, { 8}, { 8}}, {{137}, {137}, { 8}, { 8}}, {{ 73}, { 73}, { 8}, { 8}}, {{201}, {201}, { 8}, { 8}},

{{ 41}, { 41}, { 8} ,{ 8}}, {{169}, {169}, { 8}, { 8}}, {{105}, {105}, { 8}, { 8}}, {{233}, {233}, { 8}, { 8}}, {{ 25}, { 25}, { 8}, { 8}},

{{153}, {153}, { 8}, { 8}}, {{ 89}, { 89}, { 8}, { 8}}, {{217}, {217}, { 8}, { 8}}, {{ 57}, { 57}, { 8}, { 8}}, {{185}, {185}, { 8}, { 8}},

{{121}, {121}, { 8}, { 8}}, {{249}, {249}, { 8}, { 8}}, {{ 5}, { 5}, { 8}, { 8}}, {{133}, {133}, { 8}, { 8}}, {{ 69}, { 69}, { 8}, { 8}},

{{197}, {197}, { 8}, { 8}}, {{ 37}, { 37}, { 8}, { 8}}, {{165}, {165}, { 8}, { 8}}, {{101}, {101}, { 8}, { 8}}, {{229}, {229}, { 8}, { 8}},

{{ 21}, { 21}, { 8}, { 8}}, {{149}, {149}, { 8}, { 8}}, {{ 85}, { 85}, { 8}, { 8}}, {{213}, {213}, { 8}, { 8}}, {{ 53}, { 53}, { 8}, { 8}},

{{181}, {181}, { 8}, { 8}}, {{117}, {117}, { 8}, { 8}}, {{245}, {245}, { 8}, { 8}}, {{ 13}, { 13}, { 8}, { 8}}, {{141}, {141}, { 8}, { 8}},

{{ 77}, { 77}, { 8}, { 8}}, {{205}, {205}, { 8}, { 8}}, {{ 45}, { 45}, { 8}, { 8}}, {{173}, {173}, { 8}, { 8}}, {{109}, {109}, { 8}, { 8}},

{{237}, {237}, { 8}, { 8}}, {{ 29}, { 29}, { 8}, { 8}}, {{157}, {157}, { 8}, { 8}}, {{ 93}, { 93}, { 8}, { 8}}, {{221}, {221}, { 8}, { 8}},

{{ 61}, { 61}, { 8}, { 8}}, {{189}, {189}, { 8}, { 8}}, {{125}, {125}, { 8}, { 8}}, {{253}, {253}, { 8}, { 8}}, {{ 19}, { 19}, { 9}, { 9}},

{{275}, {275}, { 9}, { 9}}, {{147}, {147}, { 9}, { 9}}, {{403}, {403}, { 9}, { 9}}, {{ 83}, { 83}, { 9}, { 9}}, {{339}, {339}, { 9}, { 9}},

{{211}, {211}, { 9}, { 9}}, {{467}, {467}, { 9}, { 9}}, {{ 51}, { 51}, { 9}, { 9}}, {{307}, {307}, { 9}, { 9}}, {{179}, {179}, { 9}, { 9}},

{{435}, {435}, { 9}, { 9}}, {{115}, {115}, { 9}, { 9}}, {{371}, {371}, { 9}, { 9}}, {{243}, {243}, { 9}, { 9}}, {{499}, {499}, { 9}, { 9}},

{{ 11}, { 11}, { 9}, { 9}}, {{267}, {267}, { 9}, { 9}}, {{139}, {139}, { 9}, { 9}}, {{395}, {395}, { 9}, { 9}}, {{ 75}, { 75}, { 9}, { 9}},

{{331}, {331}, { 9}, { 9}}, {{203}, {203}, { 9}, { 9}}, {{459}, {459}, { 9}, { 9}}, {{ 43}, { 43}, { 9}, { 9}}, {{299}, {299}, { 9}, { 9}},

{{171}, {171}, { 9}, { 9}}, {{427}, {427}, { 9}, { 9}}, {{107}, {107}, { 9}, { 9}}, {{363}, {363}, { 9}, { 9}}, {{235}, {235}, { 9}, { 9}},

{{491}, {491}, { 9}, { 9}}, {{ 27}, { 27}, { 9}, { 9}}, {{283}, {283}, { 9}, { 9}}, {{155}, {155}, { 9}, { 9}}, {{411}, {411}, { 9}, { 9}},

{{ 91}, { 91}, { 9}, { 9}}, {{347}, {347}, { 9}, { 9}}, {{219}, {219}, { 9}, { 9}}, {{475}, {475}, { 9}, { 9}}, {{ 59}, { 59}, { 9}, { 9}},

{{315}, {315}, { 9}, { 9}}, {{187}, {187}, { 9}, { 9}}, {{443}, {443}, { 9}, { 9}}, {{123}, {123}, { 9}, { 9}}, {{379}, {379}, { 9}, { 9}},

{{251}, {251}, { 9}, { 9}}, {{507}, {507}, { 9}, { 9}}, {{ 7}, { 7}, { 9}, { 9}}, {{263}, {263}, { 9}, { 9}}, {{135}, {135}, { 9}, { 9}},

{{391}, {391}, { 9}, { 9}}, {{ 71}, { 71}, { 9}, { 9}}, {{327}, {327}, { 9}, { 9}}, {{199}, {199}, { 9}, { 9}}, {{455}, {455}, { 9}, { 9}},

{{ 39}, { 39}, { 9}, { 9}}, {{295}, {295}, { 9}, { 9}}, {{167}, {167}, { 9}, { 9}}, {{423}, {423}, { 9}, { 9}}, {{103}, {103}, { 9}, { 9}},

{{359}, {359}, { 9}, { 9}}, {{231}, {231}, { 9}, { 9}}, {{487}, {487}, { 9}, { 9}}, {{ 23}, { 23}, { 9}, { 9}}, {{279}, {279}, { 9}, { 9}},

{{151}, {151}, { 9}, { 9}}, {{407}, {407}, { 9}, { 9}}, {{ 87}, { 87}, { 9}, { 9}}, {{343}, {343}, { 9}, { 9}}, {{215}, {215}, { 9}, { 9}},

{{471}, {471}, { 9}, { 9}}, {{ 55}, { 55}, { 9}, { 9}}, {{311}, {311}, { 9}, { 9}}, {{183}, {183}, { 9}, { 9}}, {{439}, {439}, { 9}, { 9}},

{{119}, {119}, { 9}, { 9}}, {{375}, {375}, { 9}, { 9}}, {{247}, {247}, { 9}, { 9}}, {{503}, {503}, { 9}, { 9}}, {{ 15}, { 15}, { 9}, { 9}},

{{271}, {271}, { 9}, { 9}}, {{143}, {143}, { 9}, { 9}}, {{399}, {399}, { 9}, { 9}}, {{ 79}, { 79}, { 9}, { 9}}, {{335}, {335}, { 9}, { 9}},

{{207}, {207}, { 9}, { 9}}, {{463}, {463}, { 9}, { 9}}, {{ 47}, { 47}, { 9}, { 9}}, {{303}, {303}, { 9}, { 9}}, {{175}, {175}, { 9}, { 9}},

{{431}, {431}, { 9}, { 9}}, {{111}, {111}, { 9}, { 9}}, {{367}, {367}, { 9}, { 9}}, {{239}, {239}, { 9}, { 9}}, {{495}, {495}, { 9}, { 9}},

{{ 31}, { 31}, { 9}, { 9}}, {{287}, {287}, { 9}, { 9}}, {{159}, {159}, { 9}, { 9}}, {{415}, {415}, { 9}, { 9}}, {{ 95}, { 95}, { 9}, { 9}},

{{351}, {351}, { 9}, { 9}}, {{223}, {223}, { 9}, { 9}}, {{479}, {479}, { 9}, { 9}}, {{ 63}, { 63}, { 9}, { 9}}, {{319}, {319}, { 9}, { 9}},

{{191}, {191}, { 9}, { 9}}, {{447}, {447}, { 9}, { 9}}, {{127}, {127}, { 9}, { 9}}, {{383}, {383}, { 9}, { 9}}, {{255}, {255}, { 9}, { 9}},

{{511}, {511}, { 9}, { 9}}, {{ 0}, { 0}, { 7}, { 7}}, {{ 64}, { 64}, { 7}, { 7}}, {{ 32}, { 32}, { 7}, { 7}}, {{ 96}, { 96}, { 7}, { 7}},

{{ 16}, { 16}, { 7}, { 7}}, {{ 80}, { 80}, { 7}, { 7}}, {{ 48}, { 48}, { 7}, { 7}}, {{112}, {112}, { 7}, { 7}}, {{ 8}, { 8}, { 7}, { 7}},

{{ 72}, { 72}, { 7}, { 7}}, {{ 40}, { 40}, { 7}, { 7}}, {{104}, {104}, { 7}, { 7}}, {{ 24}, { 24}, { 7}, { 7}}, {{ 88}, { 88}, { 7}, { 7}},

{{ 56}, { 56}, { 7}, { 7}}, {{120}, {120}, { 7}, { 7}}, {{ 4}, { 4}, { 7}, { 7}}, {{ 68}, { 68}, { 7}, { 7}}, {{ 36}, { 36}, { 7}, { 7}},

{{100}, {100}, { 7}, { 7}}, {{ 20}, { 20}, { 7}, { 7}}, {{ 84}, { 84}, { 7}, { 7}}, {{ 52}, { 52}, { 7}, { 7}}, {{116}, {116}, { 7}, { 7}},

{{ 3}, { 3}, { 8}, { 8}}, {{131}, {131}, { 8}, { 8}}, {{ 67}, { 67}, { 8}, { 8}}, {{195}, {195}, { 8}, { 8}}, {{ 35}, { 35}, { 8}, { 8}},

{{163}, {163} ,{ 8}, { 8}}, {{ 99}, { 99}, { 8}, { 8}}, {{227}, {227}, { 8}, { 8}}

};

// дадатковая колькасць бітоў для сівалаў/длін, дыстанцый і длін кадоў

const int extra\_lbits[LENGTH\_CODES]

= { 0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,0 };

const int extra\_dbits[D\_CODES]

= { 0,0,0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11,12,12,13,13 };

const int extra\_blbits[BL\_CODES]

= { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3,7 };

const unsigned short bl\_order[BL\_CODES]

= { 16,17,18,0,8,7,9,6,10,5,11,4,12,3,13,2,14,1,15 };

// апісанне для статычных табліц сімвалаў/длін, дыстанцый і длін кадоў

const StaticTreeDesc static\_l\_desc =

{ static\_ltree, extra\_lbits, LITERALS + 1, L\_CODES, MAX\_BITS };

const StaticTreeDesc static\_d\_desc =

{ static\_dtree, extra\_dbits, 0, D\_CODES, MAX\_BITS };

const StaticTreeDesc static\_bl\_desc =

{ (const huffmanTable\*)0, extra\_blbits, 0, BL\_CODES, MAX\_BL\_BITS };

class Encoder : public ICoder

{

private:

unsigned char\* \_uStr = new unsigned char[16384];// радок, у каторы запісваецца інфармацыя з файла

unsigned int \_len; // дліна \_uStr

std::string \_strout;

std::string \_encodeStr; // радок з сціснутай інфармацыя

unsigned short\* \_head = new unsigned short[52685] { 0 }; // хэш-табліца

unsigned short\* \_prev = new unsigned short[52685] { 52685 }; // хэш-табліца

int \_index = 0; // бягучы індэкс радка \_uStr

huffmanTable \_dynamicLiteralTree[573]; // табліца для сімвалаў/длін

huffmanTable \_dynamicDistTree[61]; // табліца для дыстанцый

huffmanTable \_bitLenTree[39]; // табліца для длін кадоў

TreeDesc \_lDesc;

TreeDesc \_distDesc;

TreeDesc \_bitLDesc;

unsigned short \_bitLenCount[16]; // масіў паўтораў длін кадоў

int \_heap[573]; // куча для будавання табліцы хафмана

int \_heapLen; // памер кучы

int \_heapMaxLen; // максімальны памер кучы

unsigned char \_depth[573] = { 'H' }; // глыбіня вузлоў табліцы

unsigned int \_hashMask = 32767; // маска для хэшырвання

unsigned int \_hashShift = 5; // зрух для хэша

unsigned int \_insertHash; // хэш

unsigned int \_winMask = 32767; // маска для акна

unsigned int \_hashHead = 0; // індэкс хэша ў табліцы

unsigned char\* \_symbolBuf = new unsigned char[3 \* 16384]; // сімвальны буфер

unsigned int \_symbolNext; // колькасць элементаў у буферы

unsigned int \_matchLen = 0; // дліна супадзення

unsigned int \_prevLen = 0; // мінулая дліна супадзення

unsigned int \_prevMatchStart = 0; // старт мінулага супадзення

unsigned int \_matchStart = 0; // старт супадзення

unsigned long \_optLen; // памер файла з дынамічнымі табліцамі

unsigned long \_staticLen; // памер файла з статычнымі табліцамі

unsigned short \_bitBuf; // бітавы буфер

int \_bitValid; // колькасць даступных бітоў

unsigned long \_adler32; // значэнне чэк-сумы

Logger \_logger;

FileContainer \_container;

File \_fileData;

public:

// канструктар

Encoder(Logger& logger, FileContainer& container)

{

\_logger = logger;

\_container = container;

}

void Compression();

// хэш-функцыя

void InsertStringToHash();

// пошук большага супадзення

unsigned int LongestMatch();

// увод блока

void InputBlock();

// выбар лепшага спосаба сціску

void DefineBestSolution();

void BuildTree(TreeDesc& desc);

int BuildBitLenTree();

// функцыя падлічэння колькасці пазтарэння длін кодаў

void ScanTree(huffmanTable\* Tree, int maxCode);

// сартыроўка кучы

void DownHeap(huffmanTable\* dynTree, int maxCode);

// ствараэнне кодаў

void GenerateCodes(huffmanTable\* dynTree, int maxCode);

// рэверсны запіс кодаў

unsigned int BitReverse(unsigned int code, int len);

// стварэнне длін кодаў

int GenerateBitLen(TreeDesc desc);

// запіс магічнай лічбы ў сціснуты радок

void SetHeader();

// запіс бітоў у сціснуты радок

void SendBits(int value, int length);

// запіс кода у сціснуты радок

void SendCode(int c, const huffmanTable\* Tree);

// запіс рэверснага байта

void PutShort(unsigned short byte);

// запіс байта

void PutByte(unsigned short byte);

// запіс значэнне чэк-сумы

void SetAdler(unsigned long bytes);

// функцыя запісу сціснутага блока

void CompressBlock(const huffmanTable\* ltree, const huffmanTable\* dtree);

// функцыя запісу дрэваў

void SendTrees(int lcodes, int dcodes, int blcodes);

// функцыя запісу дрэва

void SendTree(huffmanTable\* Tree, int maxCode);

};

\\ файл “Encoder.cpp”

#include <iostream>

#include <string>

#include "Exceptions.h"

#include "Encoder.h"

void Encoder::Compression()

{

bool matchAvailable = false;

\_len = ReadFromFile(\_container, \_uStr);

\_fileData = \_container.GetTop()->data;

\_insertHash = \_uStr[0];

\_adler32 = Adler32(\_uStr, 1, \_len); // вызначаем чэк-суму

\_insertHash = (((\_insertHash << \_hashShift) ^ (\_uStr[1])) & \_hashMask); // фармуем хэш

while (1)

{

\_hashHead = 0;

if (\_index + 2 < \_len)

{

InsertStringToHash();

}

// запіс новых дліну і індэкс супадзення

\_prevLen = \_matchLen;

\_prevMatchStart = \_matchStart;

\_matchLen = 2;

if (\_hashHead != 0 && \_index - \_hashHead <= MAX\_DIST(\_len) && \_prevLen < 16)

{

\_matchLen = LongestMatch();

if (\_matchLen == MIN\_MATCH && (\_index - \_matchStart) > 4096)

{

\_matchLen = MIN\_MATCH - 1;

}

}

if (\_prevLen >= 3 && \_matchLen <= \_prevLen)

{

unsigned int distance = \_index - \_prevMatchStart - 1; // дыстанцыя да пачатку супайшага радка

unsigned int length = \_prevLen - 3; // дліна счытвання

// запіс у сімвальны буфер

\_symbolBuf[\_symbolNext++] = (unsigned char)(distance);

\_symbolBuf[\_symbolNext++] = (unsigned char)(distance >> 8);

\_symbolBuf[\_symbolNext++] = (unsigned char)length;

\_strout += "(";

\_strout += std::to\_string(distance);

\_strout += ", ";

\_strout += std::to\_string(distance >> 8);

\_strout += ", ";

\_strout += std::to\_string(length);

\_strout += ")";

distance--;

// запіс у дынамічныя дрэвы

\_dynamicLiteralTree[\_length\_code[length] + 256 + 1].frequence++;

\_dynamicDistTree[d\_code(distance)].frequence++;

\_prevLen -= 2;

do

{

++\_index;

if (\_index + 2 < \_len)

{

InsertStringToHash();

}

} while (--\_prevLen != 0);

matchAvailable = false;

\_matchLen = 2;

\_index++;

}

else if (matchAvailable != false)

{

// запіс у буфер

\_symbolBuf[\_symbolNext++] = 0;

\_symbolBuf[\_symbolNext++] = 0;

\_symbolBuf[\_symbolNext++] = \_uStr[\_index - 1];

\_strout += "(0, 0, ";

\_strout += \_uStr[\_index - 1];

\_strout += ")";

\_dynamicLiteralTree[\_uStr[\_index - 1]].frequence++;

\_index++;

}

else

{

matchAvailable = true;

\_index++;

}

if (\_index == \_len)

{

break;

}

}

// канчатковы запіс у буфер

if (matchAvailable)

{

// запіс у буфер

\_symbolBuf[\_symbolNext++] = 0;

\_symbolBuf[\_symbolNext++] = 0;

\_symbolBuf[\_symbolNext++] = \_uStr[\_index - 1];

\_strout += "(0, 0, ";

\_strout += \_uStr[\_index - 1];

\_strout += ")";

\_dynamicLiteralTree[\_uStr[\_index - 1]].frequence++;

}

InputBlock();

}

void Encoder::InsertStringToHash()

{

\_insertHash = (((\_insertHash << \_hashShift) ^ (\_uStr[\_index + 2])) & \_hashMask);

\_hashHead = \_prev[\_index & \_winMask] = \_head[\_insertHash]; // індэкс мінулага радка з такім хэшам

\_head[\_insertHash] = (unsigned short)\_index; // новы індэкс

}

unsigned int Encoder::LongestMatch()

{

unsigned int limit = \_index > MAX\_DIST(\_len) ? \_index - MAX\_DIST(\_len) : 0; // вызначаем мінімальны індэкс для індэкса супадзення

unsigned char\* currStr = \_uStr + \_index; // бягучы радок

unsigned int currMatchPos = \_hashHead; // індэкс супадзення

int currMatchLen = 0; // дліна супадзення

int bestPrevLen = \_prevLen;

int matchLen = 0; // дліна супадзення

int indexOfprev = 0; // індэкс мінулага супадзення

int niceMatch = 128; // дліна супадзення, пасля каторай спыняюцца пошукі супадзенняў

unsigned char prevMatchLastChar = currStr[bestPrevLen];

unsigned char prevMatchPreLastChar = currStr[bestPrevLen - 1];

unsigned char\* maxMatchStr = \_uStr + \_index + 258; // максімальны радок супадзення

\_matchLen = 0;

do

{

matchLen = 0;

unsigned char\* match = \_uStr + currMatchPos; // супаўшы радок

\_hashHead = currMatchPos;

// праверка ці роўны два першыя і апошнія элементы знойдзеных радкоў

if (match[bestPrevLen] != prevMatchLastChar ||

match[bestPrevLen - 1] != prevMatchPreLastChar ||

match[0] != currStr[0] ||

match[1] != currStr[1])

{

// індэкс пачатка мінулага радка з такім жа хэшам

indexOfprev = \_prev[currMatchPos & \_winMask];

currMatchPos = indexOfprev;

continue;

}

int i = 0;

// падлічваем дліну супадзення

while (currStr != 0)

{

if ((match[i] != currStr[i] && currStr[i] != 0) || (match >= maxMatchStr) || matchLen == 258)

{

break;

}

i++;

matchLen++;

}

if (matchLen > bestPrevLen)

{

bestPrevLen = matchLen;

\_matchStart = \_hashHead;

if (bestPrevLen >= niceMatch)

{

break;

}

prevMatchLastChar = currStr[bestPrevLen];

prevMatchPreLastChar = currStr[bestPrevLen - 1];

}

indexOfprev = \_prev[currMatchPos & \_winMask];

currMatchPos = indexOfprev;

} while (currMatchPos > limit);

return bestPrevLen;

}

void Encoder::InputBlock()

{

// ініцыалізацыя табліц

\_dynamicLiteralTree[256].frequence = 1;

\_lDesc.dynamicTree = \_dynamicLiteralTree;

\_lDesc.statDesc = &static\_l\_desc;

\_distDesc.dynamicTree = \_dynamicDistTree;

\_distDesc.statDesc = &static\_d\_desc;

\_bitLDesc.dynamicTree = \_bitLenTree;

\_bitLDesc.statDesc = &static\_bl\_desc;

DefineBestSolution();

WriteToFile(\_container, \_encodeStr);

}

void Encoder::DefineBestSolution()

{

unsigned long optLen, staticLen;

int maxBitLenInd = 0;

BuildTree(\_lDesc);

BuildTree(\_distDesc);

maxBitLenInd = BuildBitLenTree();

// падлік памераў файла

optLen = (\_optLen + 10) >> 3;

staticLen = (\_staticLen + 10) >> 3;

if (staticLen <= optLen)

{

SetHeader();

SendBits(((STATIC\_TREES << 1) + 1), 3);

CompressBlock(static\_ltree, static\_dtree);

}

else

{

SetHeader();

SendBits((DYN\_TREES << 1) + 1, 3);

SendTrees(\_lDesc.maxCode + 1, \_distDesc.maxCode + 1, maxBitLenInd + 1);

CompressBlock(\_dynamicLiteralTree, \_dynamicDistTree);

}

// дазапіс застаўшыхся бітоў

if (\_bitValid > 8)

{

PutShort(\_bitBuf);

}

else if (\_bitValid > 0)

{

PutByte(\_bitBuf);

}

\_bitBuf = 0;

\_bitValid = 0;

// запіс чэк-сумы

SetAdler(\_adler32 >> 16);

SetAdler(\_adler32 & 0xffff);

}

void Encoder::BuildTree(TreeDesc& desc)

{

huffmanTable\* dynTree = desc.dynamicTree; // дынамічнае дрэва

const huffmanTable\* staticTree = desc.statDesc->staticTree; // статычнае дрэва

int elems = desc.statDesc->elems; // элементы

int maxCode = -1; // апошні сустрэчаны код

int dadNode; // бацькоўскі вузел

\_heapLen = 0;

\_heapMaxLen = 573;

std::fill(\_depth, \_depth + sizeof(\_depth), 'H'); // запоўненне "Н"

// пошук maxCode, падлік дліны кучы і запіс яго элементаў

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

{

if (dynTree[i].frequence != 0)

{

\_heap[++\_heapLen] = maxCode = i;

\_depth[i] = 0;

}

else

{

dynTree[i].len = 0;

}

}

desc.maxCode = maxCode;

int i = 0;

// частковая сартыроўка кучы

for (i = \_heapLen / 2; i >= 1; i--)

{

DownHeap(dynTree, i);

}

int firstNode = 0;

int secondNode = 0;

dadNode = elems;

// поўная сартыроўка кучы з фармаваннем вузлоў

do

{

firstNode = \_heap[1];

\_heap[1] = \_heap[\_heapLen--];

DownHeap(dynTree, 1);

secondNode = \_heap[1];

\_heap[--\_heapMaxLen] = firstNode;

\_heap[--\_heapMaxLen] = secondNode;

dynTree[dadNode].frequence = dynTree[firstNode].frequence + dynTree[secondNode].frequence;

\_depth[dadNode] = (unsigned short)((\_depth[firstNode] >= \_depth[secondNode] ? \_depth[firstNode] : \_depth[secondNode]) + 1);

dynTree[firstNode].dad = dynTree[secondNode].dad = dadNode;

\_heap[1] = dadNode++;

DownHeap(dynTree, 1);

} while (\_heapLen >= 2);

\_heap[--\_heapMaxLen] = \_heap[1];

GenerateBitLen(desc);

GenerateCodes(dynTree, maxCode);

}

void Encoder::DownHeap(huffmanTable\* dynTree, int i)

{

int heapElem = \_heap[i];

int j = i << 1;

while (j <= \_heapLen)

{

if (j < \_heapLen && smaller(dynTree, \_heap[j + 1], \_heap[j], \_depth))// svaller выконваецца, калі дліна вузла j менш за дліну па j + 1

{

j++;

}

if (smaller(dynTree, heapElem, \_heap[j], \_depth)) // выконваецца, калі дліна вузла j менш за дліну па heapElem

{

break;

}

\_heap[i] = \_heap[j];

i = j;

j <<= 1;

}

\_heap[i] = heapElem;

}

int Encoder::GenerateBitLen(TreeDesc desc)

{

huffmanTable\* dynTree = desc.dynamicTree;

int maxCode = desc.maxCode;

const huffmanTable\* staticTree = desc.statDesc->staticTree;

const int\* extraBits = desc.statDesc->extraBits;

int base = desc.statDesc->extraBase;

int maxBitLen = desc.statDesc->maxBitLen;

int heapInd = 0; // індэкс кучы

int n = 0; // індэкс

int bitsAmount = 0; // колькасць бітоў

int xBits = 0; // колькасць дад. бітоў

unsigned short freq = 0; // частата сустракання

for (bitsAmount = 0; bitsAmount <= MAX\_BITS; bitsAmount++)

{

\_bitLenCount[bitsAmount] = 0;

}

dynTree[\_heap[\_heapMaxLen]].len = 0;

// фармаванне длін бітоў

for (heapInd = \_heapMaxLen + 1; heapInd < HEAP\_SIZE; heapInd++)

{

n = \_heap[heapInd];

bitsAmount = dynTree[dynTree[n].dad].len + 1;

dynTree[n].len = bitsAmount;

if (n > maxCode) // пакуль не звычайны вузел

{

continue;

}

\_bitLenCount[bitsAmount]++;

xBits = 0;

if (n >= base)

{

xBits = extraBits[n - base]; // колькасць дад. бітоў

}

freq = dynTree[n].frequence;

\_optLen += freq \* (bitsAmount + xBits); // запіс колькасці выкарыставанага месца для пэўнага сімвала

if (staticTree)

{

\_staticLen += freq \* (staticTree[n].len + xBits);

}

}

return 0;

}

void Encoder::GenerateCodes(huffmanTable\* dynTree, int maxCode)

{

unsigned short nextCode[MAX\_BITS + 1];

unsigned int code = 0;

int bitInd = 0;

int n = 0;

// фармаванне кадоў

for (bitInd = 1; bitInd <= MAX\_BITS; bitInd++)

{

code = (code + \_bitLenCount[bitInd - 1]) << 1;

nextCode[bitInd] = code;

}

// "пераварочванне" кодаў

for (n = 0; n <= maxCode; n++)

{

int len = dynTree[n].len;

if (len == 0)

{

continue;

}

dynTree[n].code = BitReverse(nextCode[len]++, len);

}

}

unsigned int Encoder::BitReverse(unsigned int code, int len)

{

unsigned int res = 0;

// "пераварочванне" кода

do

{

res |= code & 1;

code >>= 1;

res <<= 1;

} while (--len > 0);

return res >> 1;

}

int Encoder::BuildBitLenTree()

{

int maxBitLenInd = 0;

ScanTree(\_dynamicLiteralTree, \_lDesc.maxCode);

ScanTree(\_dynamicDistTree, \_distDesc.maxCode);

BuildTree(\_bitLDesc);

// праверка на колькасць выкарыставаных длін кодаў

for (maxBitLenInd = BL\_CODES - 1; maxBitLenInd >= 3; maxBitLenInd--)

{

if (\_bitLenTree[bl\_order[maxBitLenInd]].len != 0)

{

break;

}

}

\_optLen += 3 \* (maxBitLenInd + 1) + 5 + 5 + 4;

return maxBitLenInd;

}

void Encoder::ScanTree(huffmanTable\* Tree, int maxCode)

{

int n; // індэкс

int prevLen = -1; // мінулая дліна

int currLen; // бягучая дліна

int nextLen = Tree[0].len; // наступная дліна

int count = 0; // колькасць паўтораных длін

int maxCount = 7; // максімальная колькасць паўтораных длін

int minCount = 4; // мінімальная колькасць паўтораных длін

if (nextLen == 0)

{

maxCount = 138;

minCount = 3;

}

Tree[maxCode + 1].len = 0xffff;

for (n = 0; n <= maxCode; n++)

{

currLen = nextLen;

nextLen = Tree[n + 1].len;

if (++count < maxCount && currLen == nextLen)

{

continue;

}

else if (count < minCount)

{

\_bitLenTree[currLen].frequence += count;

\_bitLenTree[currLen].code += count;

}

else if (currLen != 0)

{

// звычайны запіс дліны

if (currLen != prevLen)

{

\_bitLenTree[currLen].frequence++;

\_bitLenTree[currLen].code++;

}

\_bitLenTree[16].frequence++;

\_bitLenTree[16].code++;

}

else if (count <= 10)

{

\_bitLenTree[17].frequence++;

\_bitLenTree[17].code++;

}

else

{

\_bitLenTree[18].frequence++;

\_bitLenTree[18].code++;

}

count = 0;

prevLen = currLen;

if (nextLen == 0)

{

maxCount = 138;

minCount = 3;

}

else if (currLen == nextLen)

{

maxCount = 6;

minCount = 3;

}

else

{

maxCount = 7;

minCount = 4;

}

}

}

void Encoder::SetHeader()

{

unsigned int header = 30876; // магічная лічба

PutByte(header >> 8);

PutByte(header & 0xff);

}

void Encoder::PutByte(unsigned short byte)

{

\_encodeStr += byte;

}

void Encoder::SendBits(int value, int length)

{

if (\_bitValid > 16 - length)

{

\_bitBuf |= value << \_bitValid;

PutShort(\_bitBuf); // запіс бітоў

// абнаўленне буфера і даступных бітоў

\_bitBuf = (unsigned short)value >> (16 - \_bitValid);

\_bitValid += length - 16;

}

else

{

// дабаўленне ў буфер новых бітоў і абноўленне колькасць даступных бітоў

\_bitBuf |= (unsigned short)value << \_bitValid;

\_bitValid += length;

}

}

void Encoder::PutShort(unsigned short byte)

{

PutByte(byte & 0xff);

PutByte(byte >> 8);

}

void Encoder::SendTrees(int lcodes, int dcodes, int blcodes)

{

int rank = 0;

SendBits(lcodes - 257, 5); // запіс колькасці длін

SendBits(dcodes - 1, 5); // запіс колькасці дыстанцый

SendBits(blcodes - 4, 4); // запіс колькасці длін кодаў

for (rank = 0; rank < blcodes; rank++)

{

SendBits(\_bitLenTree[bl\_order[rank]].len, 3);

}

SendTree(\_dynamicLiteralTree, lcodes - 1);

SendTree(\_dynamicDistTree, dcodes - 1);

}

void Encoder::SendTree(huffmanTable\* Tree, int maxCode)

{

int n; // індэкс

int prevLen = -1; // мінулая дліна

int currLen; // бягучая дліна

int nextLen = Tree[0].len; // наступная дліна

int count = 0; // колькасць паўторанай длін

int maxCount = 7; // максімальная колькасць паўтораных длін

int minCount = 4; // мінімальная колькасць паўтораных длін

if (nextLen == 0)

{

maxCount = 138;

minCount = 3;

}

for (n = 0; n <= maxCode; n++)

{

currLen = nextLen;

nextLen = Tree[n + 1].len;

if (++count < maxCount && currLen == nextLen)

{

continue;

}

else if (count < minCount)

{

// запіс дліны

do

{

SendCode(currLen, \_bitLenTree);

} while (--count != 0);

}

else if (currLen != 0)

{

// запіс дліны

if (currLen != prevLen)

{

SendCode(currLen, \_bitLenTree);

count--;

}

SendCode(16, \_bitLenTree);

SendBits(count - 3, 2);

}

else if (count <= 10)

{

SendCode(17, \_bitLenTree);

SendBits(count - 3, 3);

}

else

{

SendCode(18, \_bitLenTree);

SendBits(count - 11, 7);

}

count = 0;

prevLen = currLen;

if (nextLen == 0)

{

maxCount = 138;

minCount = 3;

}

else if (currLen == nextLen)

{

maxCount = 6;

minCount = 3;

}

else

{

maxCount = 7;

minCount = 4;

}

}

}

void Encoder::CompressBlock(const huffmanTable\* ltree, const huffmanTable\* dtree)

{

unsigned int dist = 0; // значэнне дыстанцыі

int lChar = 0; // значэнне сімвала/дліны

unsigned int sbInd = 0; // індэкс буфера сімвалаў

unsigned int code = 0; // код

int extra = 0; // дад. біты

if (\_symbolNext != 0)

{

do

{

dist = \_symbolBuf[sbInd++] & 0xff;

dist += (unsigned int)(\_symbolBuf[sbInd++] & 0xff) << 8; // дабаўленне дад. дыстанцыі

lChar = \_symbolBuf[sbInd++];

if (dist == 0)

{

// запіс літары

SendCode(lChar, ltree);

}

else

{

// запіс дліны

code = \_length\_code[lChar];

SendCode(code + LITERALS + 1, ltree);

extra = extra\_lbits[code];

if (extra != 0)

{

// дазапіс дад. бітоў дліны

lChar -= base\_length[code];

SendBits(lChar, extra);

}

dist--;

// запіс дыстанцыі

code = (dist) < 256 ? \_dist\_code[dist] : \_dist\_code[256 + ((dist) >> 7)];

SendCode(code, dtree);

extra = extra\_dbits[code];

if (extra != 0)

{

// дазапіс дад. бітоў дыстанцыі

dist -= base\_dist[code];

SendBits(dist, extra);

}

}

} while (sbInd < \_symbolNext);

}

// запіс сімвала канца блока

SendCode(256, ltree);

}

void Encoder::SendCode(int c, const huffmanTable\* Tree)

{

SendBits(Tree[c].code, Tree[c].len);

}

void Encoder::SetAdler(unsigned long bytes)

{

PutByte(bytes >> 8);

PutByte(bytes & 0xff);

}

\\ файл “Decoder.h”

#pragma once

#include "ICoder.h"

#include "Container.h"

#include "ILogger.h"

struct code

{

unsigned char indicator; // індыкатар для вызначэння аперацыі

unsigned char bits; // біты

unsigned short value; // значэнне

};

// табліцы для сімвалаў/длін і дыстанцый

static const code lenfix[512] = {

{96,7,0},{0,8,80},{0,8,16},{20,8,115},{18,7,31},{0,8,112},{0,8,48},

{0,9,192},{16,7,10},{0,8,96},{0,8,32},{0,9,160},{0,8,0},{0,8,128},

{0,8,64},{0,9,224},{16,7,6},{0,8,88},{0,8,24},{0,9,144},{19,7,59},

{0,8,120},{0,8,56},{0,9,208},{17,7,17},{0,8,104},{0,8,40},{0,9,176},

{0,8,8},{0,8,136},{0,8,72},{0,9,240},{16,7,4},{0,8,84},{0,8,20},

{21,8,227},{19,7,43},{0,8,116},{0,8,52},{0,9,200},{17,7,13},{0,8,100},

{0,8,36},{0,9,168},{0,8,4},{0,8,132},{0,8,68},{0,9,232},{16,7,8},

{0,8,92},{0,8,28},{0,9,152},{20,7,83},{0,8,124},{0,8,60},{0,9,216},

{18,7,23},{0,8,108},{0,8,44},{0,9,184},{0,8,12},{0,8,140},{0,8,76},

{0,9,248},{16,7,3},{0,8,82},{0,8,18},{21,8,163},{19,7,35},{0,8,114},

{0,8,50},{0,9,196},{17,7,11},{0,8,98},{0,8,34},{0,9,164},{0,8,2},

{0,8,130},{0,8,66},{0,9,228},{16,7,7},{0,8,90},{0,8,26},{0,9,148},

{20,7,67},{0,8,122},{0,8,58},{0,9,212},{18,7,19},{0,8,106},{0,8,42},

{0,9,180},{0,8,10},{0,8,138},{0,8,74},{0,9,244},{16,7,5},{0,8,86},

{0,8,22},{64,8,0},{19,7,51},{0,8,118},{0,8,54},{0,9,204},{17,7,15},

{0,8,102},{0,8,38},{0,9,172},{0,8,6},{0,8,134},{0,8,70},{0,9,236},

{16,7,9},{0,8,94},{0,8,30},{0,9,156},{20,7,99},{0,8,126},{0,8,62},

{0,9,220},{18,7,27},{0,8,110},{0,8,46},{0,9,188},{0,8,14},{0,8,142},

{0,8,78},{0,9,252},{96,7,0},{0,8,81},{0,8,17},{21,8,131},{18,7,31},

{0,8,113},{0,8,49},{0,9,194},{16,7,10},{0,8,97},{0,8,33},{0,9,162},

{0,8,1},{0,8,129},{0,8,65},{0,9,226},{16,7,6},{0,8,89},{0,8,25},

{0,9,146},{19,7,59},{0,8,121},{0,8,57},{0,9,210},{17,7,17},{0,8,105},

{0,8,41},{0,9,178},{0,8,9},{0,8,137},{0,8,73},{0,9,242},{16,7,4},

{0,8,85},{0,8,21},{16,8,258},{19,7,43},{0,8,117},{0,8,53},{0,9,202},

{17,7,13},{0,8,101},{0,8,37},{0,9,170},{0,8,5},{0,8,133},{0,8,69},

{0,9,234},{16,7,8},{0,8,93},{0,8,29},{0,9,154},{20,7,83},{0,8,125},

{0,8,61},{0,9,218},{18,7,23},{0,8,109},{0,8,45},{0,9,186},{0,8,13},

{0,8,141},{0,8,77},{0,9,250},{16,7,3},{0,8,83},{0,8,19},{21,8,195},

{19,7,35},{0,8,115},{0,8,51},{0,9,198},{17,7,11},{0,8,99},{0,8,35},

{0,9,166},{0,8,3},{0,8,131},{0,8,67},{0,9,230},{16,7,7},{0,8,91},

{0,8,27},{0,9,150},{20,7,67},{0,8,123},{0,8,59},{0,9,214},{18,7,19},

{0,8,107},{0,8,43},{0,9,182},{0,8,11},{0,8,139},{0,8,75},{0,9,246},

{16,7,5},{0,8,87},{0,8,23},{64,8,0},{19,7,51},{0,8,119},{0,8,55},

{0,9,206},{17,7,15},{0,8,103},{0,8,39},{0,9,174},{0,8,7},{0,8,135},

{0,8,71},{0,9,238},{16,7,9},{0,8,95},{0,8,31},{0,9,158},{20,7,99},

{0,8,127},{0,8,63},{0,9,222},{18,7,27},{0,8,111},{0,8,47},{0,9,190},

{0,8,15},{0,8,143},{0,8,79},{0,9,254},{96,7,0},{0,8,80},{0,8,16},

{20,8,115},{18,7,31},{0,8,112},{0,8,48},{0,9,193},{16,7,10},{0,8,96},

{0,8,32},{0,9,161},{0,8,0},{0,8,128},{0,8,64},{0,9,225},{16,7,6},

{0,8,88},{0,8,24},{0,9,145},{19,7,59},{0,8,120},{0,8,56},{0,9,209},

{17,7,17},{0,8,104},{0,8,40},{0,9,177},{0,8,8},{0,8,136},{0,8,72},

{0,9,241},{16,7,4},{0,8,84},{0,8,20},{21,8,227},{19,7,43},{0,8,116},

{0,8,52},{0,9,201},{17,7,13},{0,8,100},{0,8,36},{0,9,169},{0,8,4},

{0,8,132},{0,8,68},{0,9,233},{16,7,8},{0,8,92},{0,8,28},{0,9,153},

{20,7,83},{0,8,124},{0,8,60},{0,9,217},{18,7,23},{0,8,108},{0,8,44},

{0,9,185},{0,8,12},{0,8,140},{0,8,76},{0,9,249},{16,7,3},{0,8,82},

{0,8,18},{21,8,163},{19,7,35},{0,8,114},{0,8,50},{0,9,197},{17,7,11},

{0,8,98},{0,8,34},{0,9,165},{0,8,2},{0,8,130},{0,8,66},{0,9,229},

{16,7,7},{0,8,90},{0,8,26},{0,9,149},{20,7,67},{0,8,122},{0,8,58},

{0,9,213},{18,7,19},{0,8,106},{0,8,42},{0,9,181},{0,8,10},{0,8,138},

{0,8,74},{0,9,245},{16,7,5},{0,8,86},{0,8,22},{64,8,0},{19,7,51},

{0,8,118},{0,8,54},{0,9,205},{17,7,15},{0,8,102},{0,8,38},{0,9,173},

{0,8,6},{0,8,134},{0,8,70},{0,9,237},{16,7,9},{0,8,94},{0,8,30},

{0,9,157},{20,7,99},{0,8,126},{0,8,62},{0,9,221},{18,7,27},{0,8,110},

{0,8,46},{0,9,189},{0,8,14},{0,8,142},{0,8,78},{0,9,253},{96,7,0},

{0,8,81},{0,8,17},{21,8,131},{18,7,31},{0,8,113},{0,8,49},{0,9,195},

{16,7,10},{0,8,97},{0,8,33},{0,9,163},{0,8,1},{0,8,129},{0,8,65},

{0,9,227},{16,7,6},{0,8,89},{0,8,25},{0,9,147},{19,7,59},{0,8,121},

{0,8,57},{0,9,211},{17,7,17},{0,8,105},{0,8,41},{0,9,179},{0,8,9},

{0,8,137},{0,8,73},{0,9,243},{16,7,4},{0,8,85},{0,8,21},{16,8,258},

{19,7,43},{0,8,117},{0,8,53},{0,9,203},{17,7,13},{0,8,101},{0,8,37},

{0,9,171},{0,8,5},{0,8,133},{0,8,69},{0,9,235},{16,7,8},{0,8,93},

{0,8,29},{0,9,155},{20,7,83},{0,8,125},{0,8,61},{0,9,219},{18,7,23},

{0,8,109},{0,8,45},{0,9,187},{0,8,13},{0,8,141},{0,8,77},{0,9,251},

{16,7,3},{0,8,83},{0,8,19},{21,8,195},{19,7,35},{0,8,115},{0,8,51},

{0,9,199},{17,7,11},{0,8,99},{0,8,35},{0,9,167},{0,8,3},{0,8,131},

{0,8,67},{0,9,231},{16,7,7},{0,8,91},{0,8,27},{0,9,151},{20,7,67},

{0,8,123},{0,8,59},{0,9,215},{18,7,19},{0,8,107},{0,8,43},{0,9,183},

{0,8,11},{0,8,139},{0,8,75},{0,9,247},{16,7,5},{0,8,87},{0,8,23},

{64,8,0},{19,7,51},{0,8,119},{0,8,55},{0,9,207},{17,7,15},{0,8,103},

{0,8,39},{0,9,175},{0,8,7},{0,8,135},{0,8,71},{0,9,239},{16,7,9},

{0,8,95},{0,8,31},{0,9,159},{20,7,99},{0,8,127},{0,8,63},{0,9,223},

{18,7,27},{0,8,111},{0,8,47},{0,9,191},{0,8,15},{0,8,143},{0,8,79},

{0,9,255}

};

static const code distfix[32] = {

{16,5,1},{23,5,257},{19,5,17},{27,5,4097},{17,5,5},{25,5,1025},

{21,5,65},{29,5,16385},{16,5,3},{24,5,513},{20,5,33},{28,5,8193},

{18,5,9},{26,5,2049},{22,5,129},{64,5,0},{16,5,2},{23,5,385},

{19,5,25},{27,5,6145},{17,5,7},{25,5,1537},{21,5,97},{29,5,24577},

{16,5,4},{24,5,769},{20,5,49},{28,5,12289},{18,5,13},{26,5,3073},

{22,5,193},{64,5,0}

};

// парадак для длін кодаў

const unsigned short codeLengthOrder[19] =

{ 16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15 };

// табліцы для дадатковых (extra) і стандартных (base) длін і дыстанцый

static const unsigned short lengthBase[31] = { /\* Length codes 257..285 base \*/

3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 17, 19, 23, 27, 31,

35, 43, 51, 59, 67, 83, 99, 115, 131, 163, 195, 227, 258, 0, 0 };

static const unsigned short lengthExtra[31] = { /\* Length codes 257..285 extra \*/

16, 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 18, 18, 18, 18,

19, 19, 19, 19, 20, 20, 20, 20, 21, 21, 21, 21, 16, 194, 65 };

static const unsigned short distBase[32] = { /\* Distance codes 0..29 base \*/

1, 2, 3, 4, 5, 7, 9, 13, 17, 25, 33, 49, 65, 97, 129, 193,

257, 385, 513, 769, 1025, 1537, 2049, 3073, 4097, 6145,

8193, 12289, 16385, 24577, 0, 0 };

static const unsigned short distExtra[32] = { /\* Distance codes 0..29 extra \*/

16, 16, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22,

23, 23, 24, 24, 25, 25, 26, 26, 27, 27,

28, 28, 29, 29, 64, 64 };

enum TYPE

{

FIXED\_CODE = 1,

DYNAMIC,

ADLER32,

END

};

enum DYNAMIC\_TABLE

{

CODES,

LENS,

DISTS

};

class Decoder : public ICoder

{

private:

unsigned char\* \_encStr = new unsigned char[16384]; // радок, у каторы запісваецца сціснутая інфармацыя з файла

unsigned char\* \_outStr = new unsigned char[16384]; // радок, у каторы запісваецца дэкампрэсаваная інфармацыя

unsigned char\* \_str = new unsigned char[16384]; // радок, каторы паказвае дэкампрэсаваную інфармацыю

unsigned char\* \_end = new unsigned char[16384]; // радок, да каторага счытваць

unsigned int \_lenOut = 0; // дліна дэкампрэсаванага радка

unsigned long \_len;

unsigned int \_availableEnc; // даступная колькасць байтаў для разархівацыі

unsigned int \_availableOut = 16384; // даступная колькасць байтаў для запіса ў файл

unsigned short \_header; // хэдэр

unsigned short \_type; // тып табліцы (1 - статычная, 2 - дынамічная)

bool \_last; // індыкатар апошняга блока

unsigned int \_readBytes; // колькасць прачытаных байтаў

unsigned int \_numOfCodeLen; // нумар дліны кодаў

unsigned long \_bitBuffer; // буфер для бітоў

unsigned int \_bitsToEnc; // колькасць бітоў для дэкампрэсіі

const code\* \_lengthCode = lenfix; // фіксаваная табліца для длін і сімвалаў

const code\* \_distCode = distfix; // фіксаваная табліца для дыстанцый

unsigned int \_lengthBits = 9; // максімальная колькасць бітоў для длін і сімвалаў

unsigned int \_distBits = 5; // колькасць бітоў для дыстанцый

unsigned char\* \_lastToRead; // радок, да якога счытваць

unsigned int \_nLen; // максімальная дліна кода сімвалаў/длін

unsigned int \_nDist; // максімальная дліна кода дыстанцый

unsigned int \_nCode; // максімальная дліна длін кода

unsigned short \_codeLength[320]; // табліца длін кадоў

unsigned short \_codeTable[288]; // табліца кадоў

code \_codes[1444]; // табліца з кадамі і змяшчэннямі

unsigned long \_adler32 = 1;

Logger \_logger;

FileContainer \_container;

File \_fileData;

public:

Decoder(Logger& logger, FileContainer& container)

{

\_logger = logger;

\_container = container;

}

void Decompression();

// праверка файлавых хэдэраў

void CheckFileHeader();

// счытаць біты

void ReadBits(unsigned int amount);

// счытаць байт

void PullByte();

// забраць біты з буфера

unsigned int GetBits(unsigned int amount);

// ачысціць буфер і вольныя біты

void ClearBitsAndBuf();

// "скінуць" прачытаныя біты

void DropBits(unsigned int amount);

// функцыя дэкампрэсіі

void Decode();

// дэкампрэсія длін і сімвалаў

void EncodeLength(const code\* table, unsigned int lenMask, unsigned int distMask);

// дэкампрэсія дыстанцый

void EncodeDist(const code\* table, unsigned int len);

// праверка на сімвал канца блока

unsigned short CheckEOB();

// "дачытванне" застаўшыхся недакампрэсаваных элементаў

void CompleteRead(code& table, code& lastTable, unsigned int length);

// функцыя "пераварочвання" значэнне чэк-сумы

unsigned long SwapAdler();

// счытванне табліцы

void ReadTable();

// стварэнне табліц хафмана

void CreateHuffmanTables();

// ставрэнне табліцы хафмана

void CreateHuffmanTable(int type, unsigned short\* lens, unsigned int codes, code\*\* TablePtr, unsigned int\* bits);

};

\\ файл “Decoder.cpp”

#include <iostream>

#include "Decoder.h"

#include "Exceptions.h"

void Decoder::Decompression()

{

\_len = \_availableEnc = ReadFromFile(\_container, \_encStr);

\_fileData = \_container.GetTop()->data;

CheckFileHeader();

while (1)

{

while (1)

{

if (\_type == FIXED\_CODE)

{

// працэс дэкадавання

if (\_availableEnc >= 6 && \_availableOut >= 258)

{

Decode();

break;

}

\_type = CheckEOB(); // праверка на сівал канца файла і дазапіс

}

// счытванне дадзеных для дынамічных табліц і іх далейшае будаванне

if (\_type == DYNAMIC)

{

ReadTable();

CreateHuffmanTables();

}

if (\_type == ADLER32)

{

ReadBits(32); // счытвенне 4 байтаў, у каторых ляжыць чэк-сума

\_adler32 = Adler32(\_str, \_adler32, \_lenOut);

if (\_adler32 != SwapAdler())

{

throw InvalidAdlerException(\_fileData.fileName);

}

\_type = END;

break;

}

}

if (\_type == END)

{

WriteToFile(\_container, (char\*)\_str);

break;

}

}

}

void Decoder::CheckFileHeader()

{

ReadBits(16); // счытванне магічнай лічбы

if (\_bitBuffer != 0x9c78)

{

throw InvalidMagicException(\_fileData.fileName);

}

ClearBitsAndBuf();

ReadBits(3); // счтыванне хэдэраў блока

\_last = GetBits(1);

DropBits(1);

\_type = GetBits(2); // від блока

DropBits(2);

if (\_type == 3)

{

throw InvalidHeaderException(\_fileData.fileName);

}

}

void Decoder::ReadBits(unsigned int amount)

{

// счытваем пакуль ёсць месца для даступных бітоў

while (\_bitsToEnc < amount)

{

PullByte();

}

}

void Decoder::PullByte()

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

\_readBytes++;

\_availableEnc--;

}

void Decoder::DropBits(unsigned int amount)

{

\_bitBuffer >>= amount;

\_bitsToEnc -= amount;

}

unsigned int Decoder::GetBits(unsigned int amount)

{

return \_bitBuffer & ((1 << amount) - 1);

}

void Decoder::ClearBitsAndBuf()

{

\_bitsToEnc = 0;

\_bitBuffer = 0;

}

void Decoder::Decode()

{

const code\* tablePtr; // табліца

unsigned int lengthMask = (1 << \_lengthBits) - 1; // маска для длін

unsigned int distMask = (1 << \_distBits) - 1; // маска для дыстанцый

\_lastToRead = \_encStr + (\_availableEnc - 5);

\_end = \_outStr + (\_availableOut - 257);

do

{

if (\_bitsToEnc < 15)

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

}

tablePtr = \_lengthCode + (\_bitBuffer & lengthMask);

EncodeLength(tablePtr, lengthMask, distMask);

} while (\_encStr < \_lastToRead && \_outStr < \_end);

unsigned int unusedBytes = \_bitsToEnc >> 3; // невыкарыставаныя байты

\_encStr -= unusedBytes;

\_bitsToEnc -= unusedBytes << 3;

\_bitBuffer &= (1 << \_bitsToEnc) - 1;

// змена колькасці байтаў даступных для дэкадавання

if (\_encStr < \_lastToRead)

{

\_availableEnc = 5 + (\_lastToRead - \_encStr);

}

else

{

\_availableEnc = 5 - (\_encStr - \_lastToRead);

}

// змена колькасці байтаў даступных для запіс у файл

if (\_outStr < \_end)

{

\_availableOut = 257 + (\_end - \_outStr);

}

else

{

\_availableOut += 257 - (\_outStr - \_end);

}

}

void Decoder::EncodeLength(const code\* table, unsigned int lenMask, unsigned int distMask)

{

unsigned int length = 0;

unsigned int bits = table->bits; // біты

unsigned int indicator = (unsigned int)table->indicator;

\_bitBuffer >>= bits;

\_bitsToEnc -= bits;

\_str = \_outStr - \_lenOut;

if (indicator == 0) // калі сімвал

{

\*\_outStr++ = (char)table->value;

\_lenOut++;

\_len--;

\_readBytes++;

}

else if (indicator & 16) // калі сустрэлася дліна

{

length = table->value; // значэнне дліны

indicator &= 15;

if (indicator)

{

if (\_bitsToEnc < indicator)

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

}

length += \_bitBuffer & ((1 << indicator) - 1); // дабаўленне дад. бітоў

\_bitBuffer >>= indicator;

\_bitsToEnc -= indicator;

}

if (\_bitsToEnc < 15)

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

}

table = \_distCode + (\_bitBuffer & distMask); // табліца для дыстанцый

EncodeDist(table, length);

}

else if ((indicator & 64) == 0)

{

table = \_lengthCode + table->value + (\_bitBuffer & ((1 << indicator) - 1)); // табліца для вялікіх длін

EncodeLength(table, lenMask, distMask);

}

}

void Decoder::EncodeDist(const code\* table, unsigned int length)

{

unsigned int bits = table->bits;

unsigned int distance = 0;

unsigned int indicator = 0;

\_bitBuffer >>= bits;

\_bitsToEnc -= bits;

indicator = table->indicator;

if (indicator & 16) // дыстанцыя

{

distance = table->value;

indicator &= 15;

if (\_bitsToEnc < indicator)

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

if (\_bitsToEnc < indicator)

{

\_bitBuffer += \*\_encStr++ << \_bitsToEnc;

\_bitsToEnc += 8;

}

}

distance += \_bitBuffer & ((1 << indicator) - 1); // дабаўленне дад. бітоў

\_bitBuffer >>= indicator;

\_bitsToEnc -= indicator;

unsigned char\* readFrom = \_outStr - distance;

// запіс супадзення

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

{

\*\_outStr++ = \*readFrom++;

\_lenOut++;

\_len--;

\_readBytes++;

}

}

else if ((indicator & 64) == 0)

{

table = \_distCode + table->value + (\_bitBuffer & ((1 << indicator) - 1)); // дабаўленне дад.бітоў для вялікіх дыстанцый

EncodeDist(table, length);

}

}

void Decoder::ReadTable()

{

ReadBits(14);

\_nLen = GetBits(5) + 257; // счытванне колькасць длін

DropBits(5);

\_nDist = GetBits(5) + 1; // счытванне колькасці дыстанцый

DropBits(5);

\_nCode = GetBits(4) + 4; // счытванне колькасці длін кодаў

DropBits(4);

}

void Decoder::CreateHuffmanTables()

{

\_readBytes = 0;

// счытванне длін кодаў

while (\_numOfCodeLen < \_nCode)

{

ReadBits(3);

\_codeLength[codeLengthOrder[\_numOfCodeLen++]] = (unsigned short)GetBits(3);

DropBits(3);

}

while (\_numOfCodeLen < 19)

{

\_codeLength[codeLengthOrder[\_numOfCodeLen++]] = 0;

}

unsigned int length = 0;

unsigned int copy = 0;

code table;

code\* next = \_codes;

\_lengthCode = next;

\_lengthBits = 7; // максімальная колькасць бітоў для дліны

CreateHuffmanTable(CODES, \_codeLength, 19, &next, &\_lengthBits);

\_numOfCodeLen = 0;

// счытванне дліны кодаў для дынамічны' табліц сімвалаў/длін і дыстанцый

while (\_numOfCodeLen < \_nLen + \_nDist)

{

while (1)

{

table = \_lengthCode[GetBits(\_lengthBits)];

if (table.bits <= \_bitsToEnc)

{

break;

}

PullByte();

}

// счытана звычайная дліна, запіс у табліцу

if (table.value < 16)

{

DropBits(table.bits);

\_codeLength[\_numOfCodeLen++] = table.value;

}

else

{

// счытана дліна з колькасцю паўтораў

if (table.value == 16) // для длін не роўных нулю

{

ReadBits(table.bits + 2);

DropBits(table.bits);

length = \_codeLength[\_numOfCodeLen - 1];

copy = 3 + GetBits(2);

DropBits(2);

}

else if (table.value == 17) // для длін роўных длін

{

ReadBits(table.bits + 3);

DropBits(table.bits);

length = 0;

copy = 3 + GetBits(3);

DropBits(3);

}

else // для длін роўных длін

{

ReadBits(table.bits + 7);

DropBits(table.bits);

length = 0;

copy = 11 + GetBits(7);

DropBits(7);

}

while (copy--) // запіс аднолькавых длін

{

\_codeLength[\_numOfCodeLen++] = (unsigned short)length;

}

}

}

next = \_codes;

\_lengthCode = next;

\_lengthBits = 9;

CreateHuffmanTable(LENS, \_codeLength, \_nLen, &next, &\_lengthBits); // будаванне табліцы для сімвалаў/длін

\_distCode = next;

\_distBits = 6;

CreateHuffmanTable(DISTS, \_codeLength + \_nLen, \_nDist, &next, &\_distBits); // будаванне табліц для дыстанцый

\_type = FIXED\_CODE;

}

void Decoder::CreateHuffmanTable(int type, unsigned short\* lens, unsigned int codes, code\*\* TablePtr, unsigned int\* bits)

{

unsigned short count[16];

unsigned short offset[16];

// абнуленне масіва

for (int i = 0; i <= 15; i++)

{

count[i] = 0;

}

// падлічванне колькасці

unsigned int symbol = 0;

for (; symbol < codes; symbol++)

{

count[lens[symbol]]++;

}

unsigned max = 15;

unsigned root = \*bits;

// пошук максімальнай дліны

for (; max >= 1; max--)

{

if (count[max] != 0)

{

break;

}

}

if (root > max)

{

root = max;

}

unsigned min;

// пошук мінімальнай дліны

for (min = 1; min < max; min++)

{

if (count[min] != 0)

{

break;

}

}

if (root < min)

{

root = min;

}

//фармаванне змяшчэнняў для далейшай сартыроўкі

unsigned int len = 0;

offset[1] = 0;

for (len = 1; len < 15; len++)

{

offset[len + 1] = offset[len] + count[len];

}

// фармаванне кодавай табліцы па ўзрастанню дліны па сімвалу

for (symbol = 0; symbol < codes; symbol++)

{

if (lens[symbol] != 0)

{

\_codeTable[offset[lens[symbol]]++] = (unsigned short)symbol;

}

}

const unsigned short\* base = nullptr, \* extra = nullptr;

unsigned int match = 0;

// ініцыалізацыя супадзенняў, дад. бітоў і базы

if (type == CODES)

{

base = extra = \_codeTable;

match = 20;

}

else if (type == LENS)

{

base = lengthBase;

extra = lengthExtra;

match = 257;

}

else if (type == DISTS)

{

base = distBase;

extra = distExtra;

match = 0;

}

code\* next = \*TablePtr;

code table;

int left = 0;

symbol = 0;

unsigned int huff = 0; // код

unsigned int startLen = min; // пачатак індэкса дліны

unsigned int currentInd = root; // бягучы індэкс

unsigned int drop = 0; // колькасць бітоў, каторыя трэба "апусціць"

unsigned int low = (unsigned)(-1); // трыгар для суб-табліцы

unsigned int usedEntry = 1 << root;

unsigned int mask = usedEntry - 1; // макса для параўнання з трыгерам

unsigned int inc = 0;

unsigned int fill = 0;

while (1)

{

table.bits = (unsigned char)(startLen - drop); // колькасць бітоў

if (\_codeTable[symbol] + 1 < match) // для сімвалаў і длін кодаў

{

table.indicator = (unsigned char)0;

table.value = \_codeTable[symbol];

}

else if (\_codeTable[symbol] >= match) // для длін і дыстанцый

{

table.indicator = (unsigned char)(extra[\_codeTable[symbol] - match]);

table.value = base[\_codeTable[symbol] - match];

}

else

{

table.indicator = (unsigned char)(32 + 64); // сімвал канец блока

table.value = 0;

}

// фармаванне і запіс кодаў ў табліцу

inc = 1 << (startLen - drop);

fill = 1 << currentInd;

min = fill;

do

{

fill -= inc;

next[(huff >> drop) + fill] = table;

} while (fill != 0);

inc = 1 << (startLen - 1);

while (huff & inc)

{

inc >>= 1;

}

if (inc != 0)

{

huff &= inc - 1;

huff += inc;

}

else

{

huff = 0;

}

// пераход да наступнага сімвала

symbol++;

// змена індэкса пачатку дліны і колькасці

if (--(count[startLen]) == 0)

{

if (startLen == max)

{

break;

}

startLen = lens[\_codeTable[symbol]];

}

// будаванне суб-табліцы

if (startLen > root && (huff & mask) != low)

{

if (drop == 0)

{

drop = root;

}

// зрух на мінімальнае змяшчэнне былой табліцы

next += min;

// вызначаем памеры наступнай табліцы

currentInd = startLen - drop;

left = (int)(1 << currentInd);

while (currentInd + drop < max)

{

left -= count[currentInd + drop];

if (left <= 0)

{

break;

}

currentInd++;

left <<= 1;

}

usedEntry += 1 << currentInd;

// пачатак суб-табліцы

low = huff & mask;

(\*TablePtr)[low].indicator = (unsigned char)currentInd;

(\*TablePtr)[low].bits = (unsigned char)root;

(\*TablePtr)[low].value = (unsigned short)(next - \*TablePtr);

}

}

\*TablePtr += usedEntry;

\*bits = root;

}

unsigned short Decoder::CheckEOB()

{

code table;

while (1)

{

table = \_lengthCode[GetBits(\_lengthBits)];

if (table.bits <= \_bitsToEnc)

{

break;

}

PullByte();

}

code lastTable;

// пошук дад. бітоў для длін

if (table.indicator && (table.indicator & 0xf0) == 0)

{

lastTable = table;

while (1)

{

table = \_lengthCode[lastTable.value + (GetBits(lastTable.bits + lastTable.indicator) >> lastTable.bits)];

if ((unsigned)(lastTable.bits + table.bits) <= \_bitsToEnc)

{

break;

}

PullByte();

}

DropBits(lastTable.bits);

}

unsigned int length = (unsigned)table.value;

DropBits(table.bits);

if (table.indicator & 32) // сімвал канца блока

{

\_bitBuffer >>= \_bitsToEnc & 7;

\_bitsToEnc -= \_bitsToEnc & 7;

return ADLER32;

}

if ((int)(table.indicator) == 0) // звычайны сімвал

{

\*\_outStr++ = (unsigned char)length;

\_availableOut--;

\_lenOut++;

return FIXED\_CODE;

}

CompleteRead(table, lastTable, length);

return FIXED\_CODE;

}

void Decoder::CompleteRead(code& table, code& lastTable, unsigned int length)

{

unsigned int extra = (unsigned)(table.indicator) & 15;

unsigned int offset = 0;

// счытванне дад. бітоў

if (extra != 0)

{

ReadBits(extra);

length += GetBits(extra);

DropBits(extra);

}

// атрыманне дыстанцыі

while (1)

{

table = \_distCode[GetBits(\_distBits)];

if ((unsigned)(table.bits) <= \_bitsToEnc)

{

break;

}

PullByte();

}

// пошук дад. бітоў

if ((table.indicator & 0xf0) == 0)

{

lastTable = table;

while (1)

{

table = \_distCode[lastTable.value + (GetBits(lastTable.bits + lastTable.indicator) >> lastTable.bits)];

if ((unsigned)(lastTable.bits + table.bits) <= \_bitsToEnc)

{

break;

}

PullByte();

}

DropBits(lastTable.bits);

}

DropBits(table.bits);

offset = table.value;

extra = (unsigned)table.indicator & 15;

unsigned char\* readFrom = 0;

unsigned int copy = 0, left = \_availableOut;

// атрыманне дад. біт дыстанцыі

if (extra != 0)

{

ReadBits(extra);

offset += GetBits(extra);

DropBits(extra);

}

// супадзенне

readFrom = \_outStr - offset;

copy = length;

if (copy > left)

{

copy = left;

}

// запіс супадзення ў выходны радок

left -= copy;

do

{

\*\_outStr++ = \*readFrom++;

\_availableOut--;

\_lenOut++;

} while (--copy);

}

unsigned long Decoder::SwapAdler()

{

return (((\_bitBuffer >> 24) & 0xff) + ((\_bitBuffer >> 8) & 0xff00) + ((\_bitBuffer & 0xff00) << 8) + ((\_bitBuffer & 0xff) << 24));

}

**ПРЫДАТАК Б***(абавязковы)*

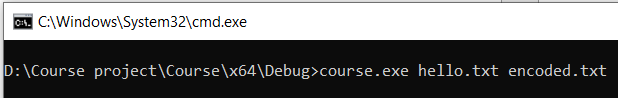
Скрыншоты працы праграмы.

Малюнак Б.1 прадстаўляе вынік праграмы без уваходных аргументаў.

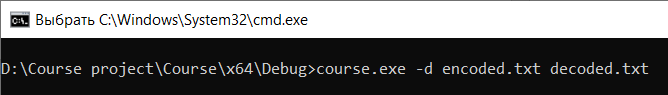


Малюнак Б.1 – прыклад праграмы пры адсутніцтве аргументаў

Малюнак Б.2 і Б.3 прадстаўляны увод для выканання кампрэсіі і дэкампрэсіі адпаведна.

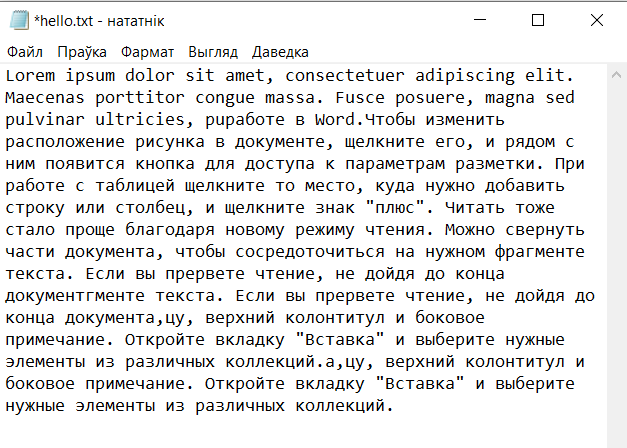


Малюнак Б.2 – прыклад увода для выканання кампрэсіі.

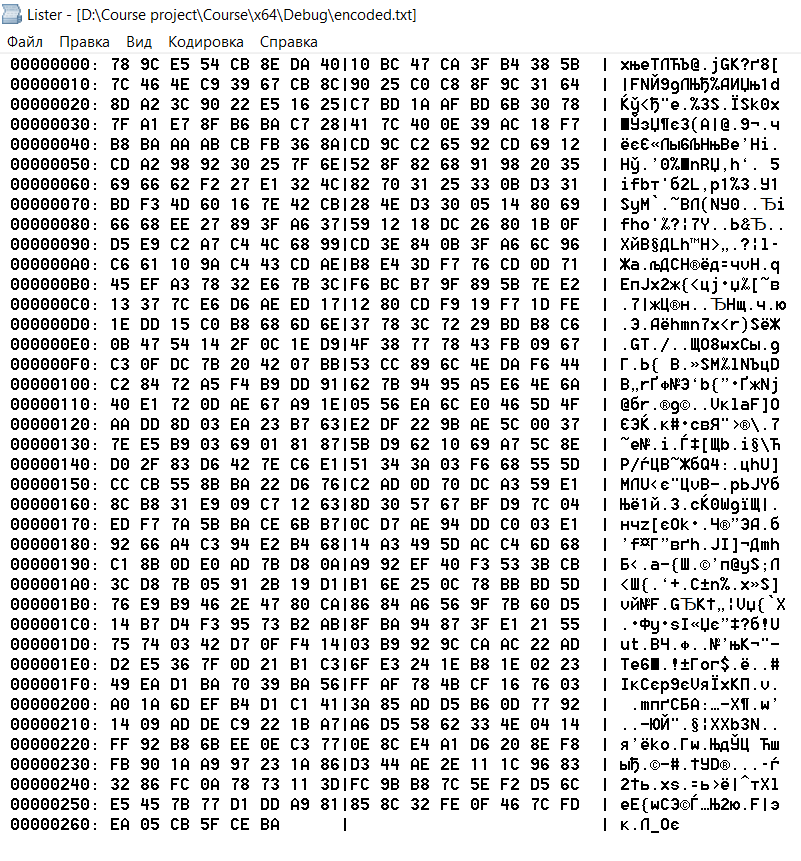


Малюнак Б.3 – прыклад увода для выканання дэкампрэсіі.

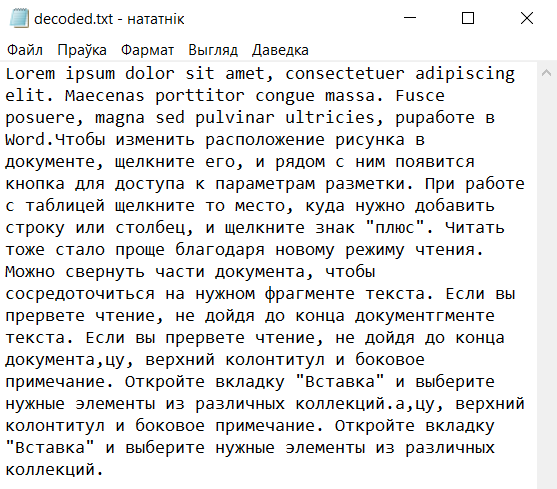
На малюнку Б.4 прадстаўлены файл да кампрэсіі, на малюнку Б.5 – пасля кампрэсіі і на малюнку Б.6 – пасля дэкампрэсіі.



Малюнак Б.4 – змест файла да кампрэсіі.

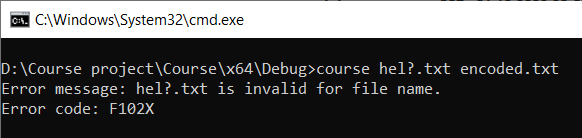


Малюнак Б.5 – змест файла пасля кампрэсіі.

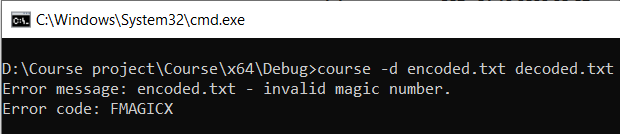


Малюнак Б.6 – змест файла пался дэкампрэсіі.

На малюнках Б.7 і Б.8 прадстаўлены дзве памылкі адпаведна: няправільна ўвод назвы файла і некарэктны фармат файла для дэкампрэсіі.



Малюнак Б.7 – вывад памылкі няправільна ўведзенага ім файла.



Малюнак Б.8 – вывад памылкі няправільнага фармата файла (некарэктны хэдэр) для сціснутага файла.

**ПРЫДАТАК В***(абавязковы)*

UML-дыяграма класаў

**ПРЫДАТАК Г**

*(абавязковы)*

Блок-схема алгарытмаў функцый Adler32() і ReadTable()

**ПРЫДАТАК Д**

(абавязковы)

Ведамасць дакументаў.