Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Ильин И. О.

Группа: М8О–206Б–19

Вариант: текстовый препроцессор

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

* Приобретение практических навыков в использовании знаний, полученных в течении курса
* Проведение исследование в выбранной предметной области

## Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Проработать полноценный текстовый препроцессор. За основу можно взять такие текстовые препроцессоры, как ed, sed и прочие. Основные особенности текстового препроцессора, которые могут быть реализованы (в зависимости от варианта):

* Транзактивность операций
* Кеширование при поиске
* Поддержка регулярных выражений
* Использование технологии MemoryMap

Вариант:

Текстовый редактор.

Запуск производится с ключом, который ограничивает количество используемой памяти (то есть, редактор загружает максимальный объем текста, но не больше указанного в ключе).

Основные особенности:

1. Редактор должен открывать и отображать файлы
2. Иметь команды добавить, удалить, изменить
3. Открывать все файлы в папке

**Общие сведения о программе**

В файле *args\_hdl.hpp* описаны функции *parse\_memsize\_num, parse\_memsize\_option* и *parse\_argv*, которые занимаются обработкой входных ключей программы. Реализации находятся в соответствующем .cpp файле.

В файле *file\_hdl.hpp* описаны функции-обретки над системными вызовами open, close, access, mmap, munmap, mremap, stat для более удобного использования.

Для работы с текстовыми файлами был реализован класс *Document*, который содержит путь до файла (*\_path*), размер загруженной части файла (*\_size*) и указатель на область памяти, куда отображается данная часть файла (*\_map*). В конструктор передается размер, который необходимо отобразить, и путь до файла (либо просто путь, если необходимо отобразить весь файл), затем, происходит открытие и отображение файла с последующим закрытием файлового дескриптора. В деструкторе производится вызовы *msync* и *munmap* для корректного сохранения изменений и уничтожения обьекта.

Данных класс содержит методы печати, вставки, удаления и изменений как символов, так и слов. Принимаются следующие общие аргументы: индекс строки и индекс символа/слова, а также длина или строка для удаления или вставки соответственно. Для работы со словами были написаны приватные методы получения начала и конца слова, что позволяет свести работу со словами к работе с символами.

Редактор предполагает два режима работы: для папок – режим чтения, для файлов – режим редактора. Для обоих режимов были реализованы соответствующие классы Reader и Editor для работы с объектами/объектом типа Document.

Класс *Reader* содержит вектор умных указателей *unique\_ptr* на объект типа Document, текущий размер всех файлов (*\_current\_size*) и верхнюю границу размера (*\_critical\_size*).

В конструкторе проверяется размер каждого файла из папки. Для очередного файла загружается максимальный объём текста. Если происходит превышение критического объема при попытке отобразить полный файл, то загружается(отображается) только его часть, равная *\_critical\_size - \_current\_size*. Класс содержит методы печати отдельных файлов, так и всех файлов.

Класс *Editor* умный указатель *unique\_ptr* на объект типа Document, текущий размер отображения(*\_current\_size*) и верхнюю границу размера (*\_critical\_size*). Имеет методы вставки, удаления и изменения. Основная функция — парсинг индексов команд и отслеживание текущего размера. Если в файл невозможно добавить строчку без переполнения, то выводится соответствующее сообщение. Также его методы содержат вызовы методов документа в блоках документа в блоках *try catch*, что позволяет использовать *throw* в методах класса *Document* без опасений того, что программа завершится.

В файле main.cpp происходит обработка входных ключей, после которой запускается либо цикл режима чтения (*reader\_loop*), либо режима редактора (*editor\_loop*).

В обоих циклах происходит создание соответствующих объектов и обработках входных команд с последующим вызовом методов.

**Основные файлы программы**

**file\_hdl.hpp:**

#pragma once

#include <string>

#include <stdexcept>

#include <errno.h>

#include <sys/mman.h>

#include <sys/fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

DIR\* open\_directory(const std::string& dir\_path);

void close\_directory(DIR\* dir);

int access\_file(const std::string& file\_path, int mode);

int open\_file(const std::string& path, int mode);

void close\_file(const int fd);

off\_t get\_file\_size(const int fd);

mode\_t get\_file\_mode(const std::string& path);

void\* map\_file\_to\_edit(int fd, size\_t size, off\_t offset);

void unmap\_file(void\* address, size\_t size);

void\* remap\_file(const char\* path, void \*old\_address, size\_t old\_size, size\_t new\_size);

**file\_hdl.cpp:**

#include "file\_hdl.hpp"

#define ERR\_ACCESS\_NO\_SUCH\_FILE 124

#define ERR\_ACCESS\_NO\_WR\_PERMS 125

#define ERR\_PERM\_DENIED 126

DIR\* open\_directory(const std::string& dir\_path) {

DIR\* to\_return;

if ((to\_return = opendir(dir\_path.data())) == NULL) {

throw std::runtime\_error("Error while opendir");

}

return to\_return;

}

void close\_directory(DIR\* dir) {

if (closedir(dir) == -1) {

throw std::runtime\_error("Error while closedir");

}

}

int open\_file(const std::string& path, int mode) {

int fd = open(path.data(), mode);

if (fd < 0) {

throw std::runtime\_error("Error while openning file");

}

return fd;

}

void close\_file(const int fd) {

if (close(fd) == -1) {

throw std::runtime\_error("Error while closing file");

}

}

void get\_file\_statbuf(const std::string& path, struct stat \* statbuf) {

int err = stat(path.data(), statbuf);

if (err < 0){

throw std::runtime\_error("Error while getting file stat");

}

}

void get\_file\_statbuf(const int fd, struct stat \* statbuf) {

int err = fstat(fd, statbuf);

if (err < 0){

throw std::runtime\_error("Error while getting file stat");

}

}

off\_t get\_file\_size(const int fd) {

struct stat statbuf;

get\_file\_statbuf(fd, &statbuf);

return statbuf.st\_size;

}

mode\_t get\_file\_mode(const std::string& path) {

struct stat statbuf;

get\_file\_statbuf(path, &statbuf);

return statbuf.st\_mode;

}

// upon success return EXIT\_SUCCESS (aka 0), either errno is returned

int access\_file(const std::string& file\_path, int mode) {

if (access(file\_path.data(), mode) == EXIT\_SUCCESS) {

return EXIT\_SUCCESS;

}

return errno;

}

void\* map\_file\_to\_edit(int fd, size\_t size, off\_t offset) {

void\* mapped = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, offset);

if (mapped == MAP\_FAILED) {

throw std::runtime\_error("Error while mmap");

}

return mapped;

}

void unmap\_file(void\* address, size\_t size) {

if (munmap(address, size) == -1) {

throw std::runtime\_error("Error while unmapping file");

}

}

void\* remap\_file(const char\* path, void\* old\_address, size\_t old\_size, size\_t new\_size) {

void\* to\_return;

#ifdef \_\_APPLE\_\_

if (old\_address != NULL) {

unmap\_file(old\_address, old\_size);

}

int fd = open\_file(path, O\_RDWR);

ftruncate(fd, new\_size);

to\_return = map\_file\_to\_edit(fd, new\_size, 0);

close\_file(fd);

return to\_return;

#else

to\_return = mremap(old\_address, old\_size, new\_size, MREMAP\_MAYMOVE);

if (to\_return == MAP\_FAILED) {

throw std::runtime\_error("Error while mremap");

}

#endif

return to\_return;

}

**args\_hdl.hpp**

#pragma once

#include <string>

#define DEFAULT\_MEMSIZE 16777216

#define ERR\_INV\_ARGS 169

void print\_usage();

unsigned long long parse\_memsize\_num(const std::string& str);

unsigned long long parse\_memsize\_option(const char\* str);

void parse\_argv(int argc, const char \* argv[], unsigned long long& memsize, std::string& path);

**args\_hdl.cpp**

#include "args\_hdl.hpp"

#include <iostream>

void print\_usage() {

std::cout << "USAGE: editor [options] [path]" << std::endl;

std::cout << "OPTIONS:" << std::endl;

std::cout << "--help Prints out the usage information." << std::endl;

std::cout << "-h Alias for --help." << std::endl;

std::cout << "--memsize=<size> Sets max size for text buffer." << std::endl;

std::cout << " Supported units: b - byte; k - kilobyte; m - megabyte."<< std::endl;

std::cout << " Default size is 16 megabytes."<< std::endl;

std::cout << "-m=<size> Alias for --memsize." << std::endl;

}

unsigned long long parse\_memsize\_num(const std::string& str) {

std::string num\_str = str.substr(0, str.size() - 1);

unsigned long long result = stol(num\_str);

if (result < 0) {

print\_usage();

exit(ERR\_INV\_ARGS);

}

switch (str[str.size() - 1]) {

case 'b':

return result;

case 'k':

return static\_cast<unsigned long long>(result) \* 1024;

case 'm':

return static\_cast<unsigned long long>(result) \* 1024 \* 1024;

default:

print\_usage();

exit(ERR\_INV\_ARGS);

}

}

// -m=9b len = 5

// --memsize=9b len = 12

unsigned long long parse\_memsize\_option(const char\* str) {

std::string option = str;

size\_t len = option.size();

std::string m\_pattern = "-m=";

std::string memsize\_pattern = "--memsize=";

if (len < 4 || (option[len-1] != 'b' && option[len-1] != 'k' && option[len-1] != 'm')) {

print\_usage();

exit(2);

}

if (option[1] == '-') {

// --memsize=<size>

if (len < 12) {

print\_usage();

exit(ERR\_INV\_ARGS);

}

if (memsize\_pattern != option.substr(0, 10)) {

print\_usage();

exit(ERR\_INV\_ARGS);

}

return parse\_memsize\_num(option.substr(10));

}

// -m=<size>

if (m\_pattern != option.substr(0, 3)) {

print\_usage();

exit(ERR\_INV\_ARGS);

}

return parse\_memsize\_num(option.substr(3));

}

// returns memsize in bytes and path from argv

void parse\_argv(int argc, const char \* argv[], unsigned long long& memsize, std::string& path) {

if (argc == 1) {

memsize = DEFAULT\_MEMSIZE;

path = "./";

} else if (argc == 2) {

if (argv[1][0] == '-') {

memsize = parse\_memsize\_option(argv[1]);

path = "./";

} else {

path = argv[1];

memsize = DEFAULT\_MEMSIZE;

}

} else if (argc == 3){

if (argv[1][0] == '-') {

memsize = parse\_memsize\_option(argv[1]);

path = argv[2];

} else if (argv[2][0] == '-') {

memsize = parse\_memsize\_option(argv[2]);

path = argv[1];

} else {

print\_usage();

exit(ERR\_INV\_ARGS);

}

} else {

print\_usage();

exit(ERR\_INV\_ARGS);

}

}

**document.hpp**

#pragma once

#include <string>

#include <iostream>

#include "file\_hdl.hpp"

class Document {

private:

std::string \_path;

unsigned long long \_size;

char\* \_map;

long long GetAbsIdx(const size\_t line\_idx, const long long idx) const;

long long FindWordStartIdx(const size\_t line\_idx, const size\_t word\_idx) const;

long long FindWordEndIdx(const size\_t line\_idx, const size\_t word\_idx) const;

size\_t CountSpaceAfter(const size\_t word\_start, const size\_t word\_len) const;

void AbsInsertAfter(const long long abs\_idx, const std::string& to\_add);

void AbsDelete(const size\_t abs\_idx, const size\_t length);

public:

Document(const std::string& file\_path);

Document(const int fd, const std::string& file\_path, unsigned long long size\_to\_load);

Document(const std::string& file\_path, unsigned long long size\_to\_load);

Document(Document&& doc) noexcept;

~Document();

unsigned long long GetSize() const;

std::string GetFileName() const;

std::string GetFullPath() const;

void Print(std::ostream& os) const;

void InsertAfter(const size\_t line\_idx, const long long idx, const std::string& to\_add);

void Delete(const size\_t line\_idx, const size\_t idx, const size\_t length);

void Edit(const size\_t line\_idx, const size\_t idx, const std::string& change\_to);

long long GetWordLength(const size\_t line\_idx, const size\_t word\_idx) const;

void InsertWordAfter(const size\_t line\_idx, const size\_t word\_idx, const std::string& word);

void DeleteWord(const size\_t line\_idx, const size\_t word\_idx);

void ChangeWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& word);

};

**document.cpp**

#include "document.hpp"

Document::Document(const std::string& file\_path)

: \_path(file\_path)

{

int fd = open\_file(file\_path, O\_RDWR);

off\_t file\_size = get\_file\_size(fd);

char\* maped\_file = (char\*)map\_file\_to\_edit(fd, file\_size, 0);

close\_file(fd);

\_size = file\_size;

\_map = maped\_file;

}

Document::Document(const std::string& file\_path, unsigned long long size\_to\_load)

: \_path(file\_path),

\_size(size\_to\_load)

{

char\* maped\_file = NULL;

if (\_size != 0) {

int fd = open\_file(file\_path, O\_RDWR);

maped\_file = (char\*)map\_file\_to\_edit(fd, \_size, 0);

close\_file(fd);

}

\_map = maped\_file;

}

Document::Document(Document&& doc) noexcept

: \_path(std::move(doc.\_path)),

\_size(std::move(doc.\_size)),

\_map(std::move(doc.\_map))

{}

Document::~Document() {

msync(\_map, \_size, MS\_SYNC);

munmap(\_map, \_size);

}

unsigned long long Document::GetSize() const {

return \_size;

}

std::string Document::GetFileName() const {

char sep = '/';

size\_t i = \_path.rfind(sep, \_path.size());

if (i != std::string::npos) {

return(\_path.substr(i + 1, \_path.size() - i));

}

return("");

}

std::string Document::GetFullPath() const {

return \_path;

}

size\_t getNumDigits(size\_t num) {

size\_t digits = 0;

while(num) {

num /= 10;

++digits;

}

return digits;

}

std::string getLinePadding(const size\_t line\_count) {

const size\_t def\_size = 3;

size\_t digits = getNumDigits(line\_count);

size\_t sp\_cnt;

if (def\_size - digits < 0) {

sp\_cnt = 0;

} else {

sp\_cnt = def\_size - digits;

}

std::string space\_before;

while (sp\_cnt > 0) {

space\_before += " ";

--sp\_cnt;

}

return "[" + space\_before + std::to\_string(line\_count) + "]";

}

void Document::Print(std::ostream& os) const {

os << "==== " << GetFileName() << " ====\n";

if (\_size == 0) {

os << "Loaded part of file is empty\n";

} else {

size\_t line\_count = 1;

os << getLinePadding(line\_count) << " ";

for (size\_t i = 0; i < \_size; ++i) {

if (\_map[i] == '\n') {

os << "\n";

++line\_count;

os << getLinePadding(line\_count) << " ";

} else {

os << \_map[i];

}

}

os << "\n";

if (\_map[\_size - 1] == '\n') {

os << "==== end ====\n";

} else {

os << "=== no endl ===\n";

}

}

}

long long Document::GetAbsIdx(const size\_t line\_idx, const long long idx) const {

// long long abs\_idx = 0; // idx after '\n'

long long abs\_idx = 0; // idx after '\n'

size\_t line\_count = 0;

while (abs\_idx < \_size) {

if (line\_count == line\_idx) { break; }

if (\_map[abs\_idx] == '\n') {

++line\_count;

}

++abs\_idx;

}

return abs\_idx + idx;

}

// insert after abs\_idx

void Document::AbsInsertAfter(const long long abs\_idx, const std::string& to\_add) {

size\_t old\_size = \_size;

size\_t new\_size = \_size + to\_add.size();

\_map = (char\*)remap\_file(\_path.data(), \_map, old\_size, new\_size);

// shift to end

for (size\_t i = new\_size - 1; i > abs\_idx + to\_add.size(); --i) {

\_map[i] = \_map[i - to\_add.size()];

}

// add text

for (size\_t i = 0; i < to\_add.size(); ++i) {

\_map[i + abs\_idx + 1] = to\_add[i];

}

\_size = new\_size;

}

void Document::InsertAfter(const size\_t line\_idx, const long long idx, const std::string& to\_add) {

long long abs\_idx = GetAbsIdx(line\_idx, idx);

if (abs\_idx != -1) {

if ((abs\_idx < -2) || abs\_idx >= \_size) {

throw std::runtime\_error("Index is out of text bounds");

}

}

AbsInsertAfter(abs\_idx, to\_add);

}

void Document::AbsDelete(const size\_t abs\_idx, const size\_t length) {

size\_t old\_size = \_size;

size\_t new\_size = \_size - length;

for (size\_t i = abs\_idx; i < new\_size; ++i) {

\_map[i] = \_map[i + length];

}

\_map = (char\*)remap\_file(\_path.data(), \_map, old\_size, new\_size);

\_size -= length;

}

void Document::Delete(const size\_t line\_idx, const size\_t idx, const size\_t length) {

long long abs\_idx = GetAbsIdx(line\_idx, idx);

if (abs\_idx < 0 || abs\_idx + length >= \_size) {

throw std::runtime\_error("Text range to delete is out of bounds");

}

AbsDelete(abs\_idx, length);

}

void Document::Edit(const size\_t line\_idx, const size\_t idx, const std::string& change\_to) {

long long abs\_idx = GetAbsIdx(line\_idx, idx);

if (abs\_idx < 0) {

throw std::runtime\_error("Start of text range to edit is out of bounds");

}

size\_t last\_idx = abs\_idx + change\_to.size();

// extend if needed

if (last\_idx >= \_size) {

size\_t change\_start\_idx = \_size - abs\_idx;

AbsInsertAfter(\_size - 1, change\_to.substr(change\_start\_idx));

}

for (size\_t i = abs\_idx; i < last\_idx; ++i) {

\_map[i] = change\_to[i - abs\_idx];

}

}

bool isSep(char cc) {

return (cc == ' ' || cc == '\t');

}

// -1 if no word at word\_idx in line

long long Document::FindWordStartIdx(const size\_t line\_idx, const size\_t word\_idx) const {

long long line\_start = GetAbsIdx(line\_idx, 0);

size\_t word\_num = word\_idx + 1;

size\_t word\_count = 0;

long long word\_start = -1;

bool in\_word = isSep(\_map[line\_start]);

for (size\_t i = line\_start; i < \_size; ++i) {

if (\_map[i] == '\n') {

break;

}

if (in\_word) {

if (isSep(\_map[i])) {

in\_word = false;

}

} else { // !in\_word

if (!isSep(\_map[i])) {

in\_word = true;

++word\_count;

if (word\_count == word\_num) {

word\_start = i;

break;

}

}

}

}

return word\_start;

}

long long Document::FindWordEndIdx(const size\_t line\_idx, const size\_t word\_idx) const {

long long word\_end = FindWordStartIdx(line\_idx, word\_idx);

if (word\_end < 0) {

return word\_end;

}

while (word\_end < \_size && \_map[word\_end] != '\n' && !isSep(\_map[word\_end])) {

++word\_end;

}

return word\_end - 1;

}

long long Document::GetWordLength(const size\_t line\_idx, const size\_t word\_idx) const {

long long word\_start = FindWordStartIdx(line\_idx, word\_idx);

if (word\_start < 0) {

return -1;

}

long long word\_end = word\_start;

while (word\_end < \_size && \_map[word\_end] != '\n' && !isSep(\_map[word\_end])) {

++word\_end;

}

--word\_end;

return word\_end - word\_start + 1;

}

void Document::InsertWordAfter(const size\_t line\_idx, const size\_t word\_idx, const std::string& word) {

long long word\_end = FindWordEndIdx(line\_idx, word\_idx);

if (word\_end < 0) {

throw std::runtime\_error("No word with such index");

}

AbsInsertAfter(word\_end, " " + word);

}

size\_t Document::CountSpaceAfter(const size\_t word\_start, const size\_t word\_len) const {

size\_t res = 0;

size\_t idx = word\_start + word\_len;

while (idx < \_size && isSep(\_map[idx])) {

++res;

}

return res;

}

void Document::DeleteWord(const size\_t line\_idx, const size\_t word\_idx) {

long long start\_idx = FindWordStartIdx(line\_idx, word\_idx);

if (start\_idx < 0) {

throw std::runtime\_error("No word with such index");

}

size\_t length = GetWordLength(line\_idx, word\_idx);

length += CountSpaceAfter(start\_idx, length);

AbsDelete(start\_idx, length);

}

void Document::ChangeWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& word) {

long long start = FindWordStartIdx(line\_idx, word\_idx);

DeleteWord(line\_idx, word\_idx);

AbsInsertAfter(start - 1, word);

}

**reader.hpp**

#pragma once

#include <vector>

#include <string>

#include <memory>

#include "document.hpp"

class Reader {

private:

const unsigned long long \_critical\_size; // in bytes

unsigned long long \_current\_size;

std::vector< std::unique\_ptr<Document> > \_docs;

public:

Reader(const std::vector<std::string>& files\_paths, const unsigned long long critical\_size);

// ~Reader();

void PrintDoc(const std::string& file\_name) const;

void PrintAll() const;

void PrintFilesList() const;

};

**reader.cpp**

#include "reader.hpp"

#include <iostream>

Reader::Reader(const std::vector<std::string>& files\_paths, const unsigned long long critical\_size)

: \_critical\_size(critical\_size),

\_current\_size(0)

{

for (auto filepath : files\_paths) {

int fd = open\_file(filepath, O\_RDWR);

off\_t file\_size = get\_file\_size(fd);

close\_file(fd);

size\_t size\_to\_load;

if (file\_size + \_current\_size > \_critical\_size) {

size\_to\_load = \_critical\_size - \_current\_size;

} else {

size\_to\_load = file\_size;

}

\_docs.push\_back(std::make\_unique<Document>(filepath, size\_to\_load));

\_current\_size += size\_to\_load;

}

}

void Reader::PrintDoc(const std::string& file\_name) const{

for (size\_t i = 0; i < \_docs.size(); ++i) {

if (\_docs[i]->GetFileName() == file\_name) {

\_docs[i]->Print(std::cout);

break;

}

if (i == \_docs.size() - 1) {

std::cout << "No such file" << std::endl;

}

}

}

void Reader::PrintAll() const {

for (size\_t i = 0; i < \_docs.size(); ++i) {

\_docs[i]->Print(std::cout);

}

}

void Reader::PrintFilesList() const {

for (size\_t i = 0; i < \_docs.size(); ++i) {

std::cout << \_docs[i]->GetFileName() << std::endl;

}

}

**editor.hpp**

#pragma once

#include <memory>

#include "document.hpp"

class Editor {

private:

const unsigned long long \_critical\_size; // in bytes

unsigned long long \_current\_size;

std::unique\_ptr<Document> \_doc;

public:

Editor(const std::string file\_path, const unsigned long long critical\_size);

void Print() const;

void Insert(const size\_t line\_idx, const size\_t idx, const std::string& to\_add);

void Delete(const size\_t line\_idx, const size\_t idx, const size\_t length);

void Edit(const size\_t line\_idx, const size\_t idx, const std::string& change\_to);

void InsertWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& to\_add);

void DeleteWord(const size\_t line\_idx, const size\_t word\_idx);

void EditWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& change\_to);

};

**editor.cpp**

#include "editor.hpp"

Editor::Editor(const std::string file\_path, const unsigned long long critical\_size)

: \_critical\_size(critical\_size),

\_current\_size(0)

{

int fd = open\_file(file\_path, O\_RDWR);

off\_t file\_size = get\_file\_size(fd);

close\_file(fd);

size\_t size\_to\_load;

if (file\_size + \_current\_size > \_critical\_size) {

size\_to\_load = \_critical\_size - \_current\_size;

} else {

size\_to\_load = file\_size;

}

\_doc = std::make\_unique<Document>(file\_path, size\_to\_load);

\_current\_size += size\_to\_load;

}

void Editor::Print() const {

\_doc->Print(std::cout);

}

// line\_idx = 1,2,... ; idx = 0,1,...

// idx = 0 to add at the beggining of the line

void Editor::Insert(const size\_t line\_idx, const size\_t idx, const std::string& to\_add) {

if (to\_add.size() + \_current\_size > \_critical\_size) {

std::cout << "Can't fit this amount of text" << std::endl;

return;

}

if (line\_idx < 1 || idx < 0) {

std::cout << "Invalid indices" << std::endl;

return;

}

unsigned long long old\_size = \_doc->GetSize();

try {

\_doc->InsertAfter(line\_idx - 1, idx - 1, to\_add);

\_current\_size += (\_doc->GetSize() - old\_size);

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

void Editor::Delete(const size\_t line\_idx, const size\_t idx, const size\_t length) {

if (line\_idx < 1 || idx < 1) {

std::cout << "Invalid indices" << std::endl;

return;

}

unsigned long long old\_size = \_doc->GetSize();

try {

\_doc->Delete(line\_idx - 1, idx - 1, length);

\_current\_size -= (old\_size - \_doc->GetSize());

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

void Editor::Edit(const size\_t line\_idx, const size\_t idx, const std::string& change\_to) {

if (line\_idx < 1 || idx < 1) {

std::cout << "Invalid indices" << std::endl;

return;

}

try {

\_doc->Edit(line\_idx - 1, idx - 1, change\_to);

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

void Editor::InsertWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& to\_add) {

if (to\_add.size() + \_current\_size > \_critical\_size) {

std::cout << "Can't fit this amount of text" << std::endl;

return;

}

if (line\_idx < 1 || word\_idx < 0) {

std::cout << "Invalid indices" << std::endl;

return;

}

unsigned long long old\_size = \_doc->GetSize();

try {

\_doc->InsertWordAfter(line\_idx - 1, word\_idx - 1, to\_add);

\_current\_size += (\_doc->GetSize() - old\_size);

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

void Editor::DeleteWord(const size\_t line\_idx, const size\_t word\_idx) {

if (line\_idx < 1 || word\_idx < 1) {

std::cout << "Invalid indices" << std::endl;

return;

}

unsigned long long old\_size = \_doc->GetSize();

try {

\_doc->DeleteWord(line\_idx - 1, word\_idx - 1);

\_current\_size -= (old\_size - \_doc->GetSize());

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

void Editor::EditWord(const size\_t line\_idx, const size\_t word\_idx, const std::string& change\_to) {

if (line\_idx < 1 || word\_idx < 1) {

std::cout << "Invalid indices" << std::endl;

return;

}

long long old\_size = \_doc->GetWordLength(line\_idx, word\_idx);

if (old\_size - change\_to.size() + \_current\_size > \_critical\_size) {

std::cout << "Can't fit this amount of text" << std::endl;

return;

}

try {

\_doc->ChangeWord(line\_idx - 1, word\_idx - 1, change\_to);

\_current\_size += old\_size - change\_to.size();

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

**main.cpp**

#include <iostream>

#include <sstream>

#include <cstring>

#include "file\_hdl.hpp"

#include "args\_hdl.hpp"

#include "reader.hpp"

#include "editor.hpp"

enum workMode {

readMode,

editMode,

unsupportedFileMode

};

void get\_files\_from\_dir(const std::string& dir\_path, std::vector<std::string>& file\_paths) {

DIR \*dir;

struct dirent \*entry;

dir = open\_directory(dir\_path);

while ((entry = readdir(dir)) != NULL) {

if (entry->d\_name[0] == '.') { continue; }

std::string full\_path = dir\_path + entry->d\_name;

if (access\_file(full\_path, R\_OK|W\_OK) == EXIT\_SUCCESS) {

file\_paths.push\_back(full\_path);

} else {

std::cerr << "Error while checking file \"" << full\_path << "\" " << strerror(errno) << std::endl;

}

}

close\_directory(dir);

}

// for regular file - editMode, for dirs - readMode, either - unsupportedFileMode

workMode pick\_work\_mode(const std::string& path) {

if (access\_file(path, F\_OK) != EXIT\_SUCCESS ) {

throw std::runtime\_error(strerror(errno));

}

mode\_t file\_type = get\_file\_mode(path);

if (S\_ISREG(file\_type)) {

return editMode;

}

if (S\_ISDIR(file\_type)) {

return readMode;

}

return unsupportedFileMode;

}

void read\_mode\_loop(const std::string& path, const unsigned long long memsize) {

std::cout << "+++ Reader mode +++" << std::endl;

try {

std::vector<std::string> file\_paths;

get\_files\_from\_dir(path, file\_paths);

Reader reader(file\_paths, memsize);

std::string cmd;

std::cout << "> ";

while(std::cin >> cmd) {

if (cmd == "print") {

std::string file\_name;

std::cin >> file\_name;

if (file\_name == "all") {

reader.PrintAll();

} else {

reader.PrintDoc(file\_name);

}

} else if (cmd == "list") {

reader.PrintFilesList();

} else if (cmd == "quit") {

break;

} else {

std::cout << "Invalid command" << std::endl;

}

std::cout << "> ";

}

std::cout << std::endl;

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

// a 4 10 add

// d 4 10

// e 4 10 change

// aw 4 5 add

// dw 4 5

// ew 4 5 change

void edit\_mode\_loop(const std::string& file\_path, const unsigned long long memsize) {

std::cout << "+++ Editor mode +++" << std::endl;

try {

Editor editor(file\_path, memsize);

std::string input\_line;

std::cout << "> ";

while(getline(std::cin, input\_line)) {

std::istringstream iss(input\_line);

// std::vector<std::string> tokens{std::istream\_iterator<std::string>{iss},

// std::istream\_iterator<std::string>{}};

// if (tokens.size() != 3 || tokens.size() != 4) {

// std::cout << "Invalid commad" << std::endl;

// }

//

std::string cmd;

iss >> cmd;

if (cmd == "print") {

editor.Print();

} else if (cmd == "a" || cmd == "aw") {

size\_t line;

size\_t idx;

std::string to\_insert;

iss >> line >> idx >> to\_insert;

if (cmd == "a")

editor.Insert(line, idx, to\_insert);

else

editor.InsertWord(line, idx, to\_insert);

} else if (cmd == "d" || cmd == "dw") {

size\_t line;

size\_t idx;

if (cmd == "d") {

size\_t length;

iss >> line >> idx >> length;

editor.Delete(line, idx, length);

} else {

iss >> line >> idx;

editor.DeleteWord(line, idx);

}

} else if (cmd == "e" || cmd == "ew") {

size\_t line;

size\_t idx;

std::string change\_to;

iss >> line >> idx >> change\_to;

if (cmd == "e")

editor.Edit(line, idx, change\_to);

else

editor.EditWord(line, idx, change\_to);

} else if (cmd == "quit") {

break;

} else {

std::cout << "Invalid command" << std::endl;

}

std::cout << "> ";

}

std::cout << "quit" << std::endl;

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

}

int main(int argc, const char \* argv[]) {

unsigned long long memsize;

std::string path;

parse\_argv(argc, argv, memsize, path);

try {

workMode mode = pick\_work\_mode(path);

if (mode == readMode) {

read\_mode\_loop(path, memsize);

} else if (mode == editMode) {

edit\_mode\_loop(path, memsize);

} else {

// unsupportedFileMode

std::cout << "Error: unsupported file. -h for help." << std::endl;

}

} catch (const std::exception& ex) {

std::cerr << ex.what() << std::endl;

}

// изменение при открытии одного файла

// если несколько - просмотр

// в какую строчку с какой позиции

return 0;

}

**Пример работы**

В первом тесте проверим работает ли ограничение для режима чтения на двух файлах, один размером 96 байт, второй 10 байт.

MacBook-Pro:test1 mr-ilin$ ls -l

total 16

-rw-r--r-- 1 mr-ilin staff 96 Jan 12 14:51 add\_test\_96b.txt

-rw-r--r-- 1 mr-ilin staff 10 Dec 30 06:33 size\_test\_10b.txt

MacBook-Pro:test1 mr-ilin$ cat add\_test\_96b.txt

012345678

012345678

012345678

012345678

012345678

012345678

012345678

012345678

012345678

012345MacBook-Pro:test1 mr-ilin$ cat size\_test\_10b.txt

234567890MacBook-Pro:test1 mr-ilin$ ../../src/my\_editor -m=100b ./

+++ Reader mode +++

> print all

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 012345678

[ 7] 012345678

[ 8] 012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

==== size\_test\_10b.txt ====

[ 1] 1234

=== no endl ===

> ^D

MacBook-Pro:test1 mr-ilin$ ../../src/my\_editor -m=10b ./

+++ Reader mode +++

> print all

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2]

==== end ====

==== size\_test\_10b.txt ====

Loaded part of file is empty

> ^D

Также проверим корректность работы программы при работе с символами.

MacBook-Pro:test1 mr-ilin$ ../../src/my\_editor -m=100b ./add\_test\_96b.txt

+++ Editor mode +++

> print

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 012345678

[ 7] 01345678

[ 8] 012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

> a 6 2 ADD

> print

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 01ADD2345678

[ 7] 01345678

[ 8] 012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

> a 8 0 A

> print

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 01ADD2345678

[ 7] 01345678

[ 8] A012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

> a 8 0 D

> a 8 0 D

Can't fit this amount of text

> print

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 01ADD2345678

[ 7] 01345678

[ 8] DA012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

> e 6 3 POW

> print

==== add\_test\_96b.txt ====

[ 1] 012345678

[ 2] 012345678

[ 3] 012345678

[ 4] 012345678

[ 5] 012345678

[ 6] 01POW2345678

[ 7] 01345678

[ 8] DA012345678

[ 9] 012345678

[ 10] 012345

=== no endl ===

Во втором тесте проверим работу со словами:

MacBook-Pro:test2 mr-ilin$ ../../src/my\_editor -m=100b ./words.txt

+++ Editor mode +++

> print

==== words.txt ====

[ 1] first second third

[ 2] change is just my word

[ 3] 123 smthg out

[ 4] lmao kek i want to break free

[ 5] its abc ADDED

=== no endl ===

> dw 3 2

> print

==== words.txt ====

[ 1] first second third

[ 2] change is just my word

[ 3] 123 out

[ 4] lmao kek i want to break free

[ 5] its abc ADDED

=== no endl ===

> aw 1 1 added

> print

==== words.txt ====

[ 1] first added second third

[ 2] change is just my word

[ 3] 123 out

[ 4] lmao kek i want to break free

[ 5] its abc ADDED

=== no endl ===

> dw 1 1

> print

==== words.txt ====

[ 1] added second third

[ 2] change is just my word

[ 3] 123 out

[ 4] lmao kek i want to break free

[ 5] its abc ADDED

=== no endl ===

**Вывод**

Для написания данного курсового проекта был выбран язык C++, поскольку посредством создания классов можно достаточно понятно и ясно разграничить ответственности отдельных модулей.

Пригодились знания, полученные в ходе работы с отображаемыми файлами. Однако, столкнулся с такой проблемой, что в Mac OS нет системного вызова remap, поэтому пришлось программировать под Ubuntu.

Вместо графического интерфейса были реализованы команды для работы со словами, а не символами.