**Створення файлового менеджеру**

*UA.IK.7116-00 01*

Курсова робота

дисципліна

**«Програмування 2. Структури даних та алгоритми»**

|  |  |
| --- | --- |
| Керівник:  Дзінько Р.І**.** | Виконав:  **Папашева А.К.**  гр. ІК-71 |

«Захист дозволено»

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2018р.

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/підпис керівника | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/підпис студента |

Зміст:

[1. Вступна частина 2](#_Toc516887600)

[2. Обґрунтування та вибір алгоритму 3](#_Toc516887601)

[3. Розробка програми 4](#_Toc516887602)

[4. Керівництво користувачу 6](#_Toc516887603)

[5. Керівництво розробнику 7](#_Toc516887604)

[6. Висновок 8](#_Toc516887605)

[7. Джерела 9](#_Toc516887606)

[Додаток 1 10](#_Toc516887607)

[Додаток 2 11](#_Toc516887608)

[Додаток 3 30](#_Toc516887609)

## Вступна частина

Файловий менеджер – це одна з найуживаніших програм на будь-якому персональному комп’ютері, вона надає користувачу зручний та швидкий спосіб переглядати файли на комп’ютері, пересувати, копіювати, видаляти і перейменовувати їх, а також створювати нові директорії, видаляти старі й реалізовувати пошук по папках.

Існує два види стандартних види файлових менеджерів:

* навігаційні
* ортодоксальні

У цій роботі було реалізовано саме ортодоксальний(двох-панельний) тип файлового менеджеру. Серед ортодоксальних файлових менеджерів для операційної системи Windows найбільш відомими є «Total Commander», «Free Commander», «FAR Manager», та інші.

Ці файлові менеджери у більшості швидші і легші(займають менше пам’яті) за їх навігаційні аналоги. Але вони мають не такий приємний інтерфейс і не є як Провідник встановленими разом із системою. Але, не зважаючи на це, багато людей, що користуються комп’ютерами у своїй професійній діяльності, віддають перевагу саме їм. Мій файловий менеджер дуже схожий на згадані вище. Але на відміну від них він використовує бібліотеку SFML для виводу графічної інформації.

## Обґрунтування та вибір алгоритму

Через те, що у цьому семестрі ми вивчали саме мову програмування C++, мною було прийнято рішення про використання саме цієї мови при написанні курсової роботи. Також мною була використана розібрана нами в ході виконання лабораторних робіт бібліотека SFML.

Для доступу до самої файлової системи було використано бібліотеку Win API.

Мною було реалізовано всі основні функції роботи з файлами, а саме:

* перехід до іншої директорії
* створення нової директорії
* копіювання директорії і файлів
* переміщення директорій і файлів
* видалення директорій і файлів
* перейменування директорій і файлів(з можливістю змінити розширення)
* пошук файлів і папок
* відкриття файлу у стандартному для нього

Для переходу до іншої директорії було використано алгоритм переліку папок і файлів представлений на сайті MSDN.

Функції копіювання, переміщення і перейменування біло реалізовано за допомогою функції SHFileOperation.

Більшість функції вдалося значно скоротити з використанням функцій Win API, а пошук був реалізовано за алгоритмом пошуку у глибину з використанням рекурсії. Але у нашому випадку, ми проходимо усі вершини графа(папки і файли) і шукаємо усі можливі варіанти збігу.

## Розробка програми

* 1. **Загальні відомості**

Файловий менеджер написаний на мові C++ за допомогою редактора VisualStudioCode, компілятора mingw32, а також утіліти make для збірки проекту.

Програма тестувалася на версіях Windows 7 і Windows 10, тож стабільність її роботи гарантується на версіях Windows 7 і вище. Для відображення російських чи українських назв файлів і папок мова системи має бути відповідною.

Для роботи папка з виконуваним файлом мають знаходитися dll-бібліотеки SFML, а також папка зі шрифтами.

* 1. **Функціональне призначення**

Файловий менеджер має надавати користувачу швидкий і зручний доступ до файлів, а також зручний інтерфейс для взаємодії з ними.

Переглядати файли у папках, відкривати їх, копіювати, перейменовувати, переміщати, видаляти і шукати.

* 1. **Опис логічної структури**

Після запуску програми виконується наступний алгоритм:

1. За допомогою функції GetLogicalDrives ми отримуємо перелік локальних дисків.
2. Відображається дві панелі з переліком дисків и блок helpbox з інструкціями до функціональних клавіш.
3. Потім програма чекає дій користувача
   1. Клавіши вверх і вниз пересувають курсор панелі і перелістують список файлів за необхідністю(вихід за висоту вікна)
   2. Клавіши вліво і вправо перемикають активну панель
   3. Клавіша «Return»(Enter) викликає перехід до папки або запускає файл у стандартній для нього програмі
   4. Функціональні клавіши(«F3», «F4», «F5», «F6», «F7», «F8») викликають відповідні дії
   5. Клавіша «Escape»(Esc) виводить у панель список локальних дисків або відміняє виконувану дію
   6. Виклик і завантаження

Для виклику програми необхідно активувати файл «main.exe» через ярлик або безпосередньо.

## Керівництво користувачу

Для встановлення файлового менеджеру треба розархівувати архів с програмою у бажану директорію і по бажанню створити ярлик виконуваного файлу на робочому столі або у меню пуск.

Якщо програму встановлено та викликано – виконати наступні дії:

1. Ознайомитися з панеллю helpbox
2. Почати роботу

Функціональні клавіши:

* F3 – створення нової директорії
* F4 – перейменування файлу або папки
* F5 – копіювання
* F6 – переміщення
* F7 – пошук у поточній папці
* F8 – видалення

### Керівництво розробнику

Файловий менеджер виконано за допомогою використання бібліотек Win API(робота з файлами), та SFML(графічний вивід і обробка подій). UML-діаграма класів(додаток 3) ілюструє структуру проекту.

Для запуску необхідні dll-бібліотеки SFML в одній директорії з виконуваним файлом, а також папка fonts з шрифтами(а саме Azbuka.ttf).

Програма містить у собі 3 функціональні класи(file\_manager, Panel, searcher).

Клас file\_manager реалізує відображення інформації у вікні її компоновку, а також функцію Navigator, що реалізовує відповідні до нажатої клавіши дії. Переключення між панелями відбувається за допомогою зміни значення поля Panel\* activePanel.

Клас Panel реалізує усі функції роботи з файлами і їх компонування для зручного відображення.

Клас searcher вивід вікна з результатами пошуку, і шляху до обраного файлу чи папки у Panel для запуску чи переходу.

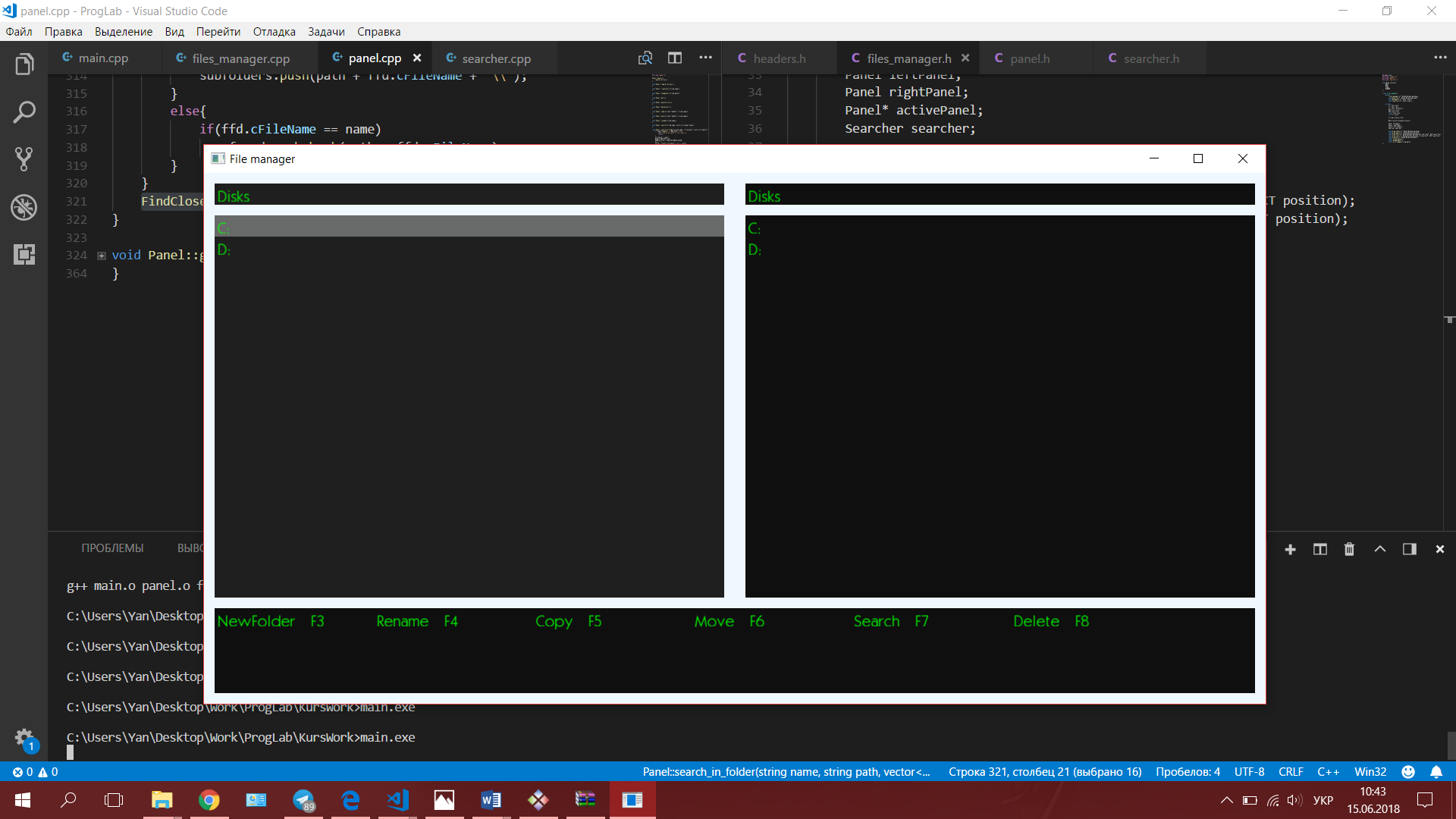
### Висновок

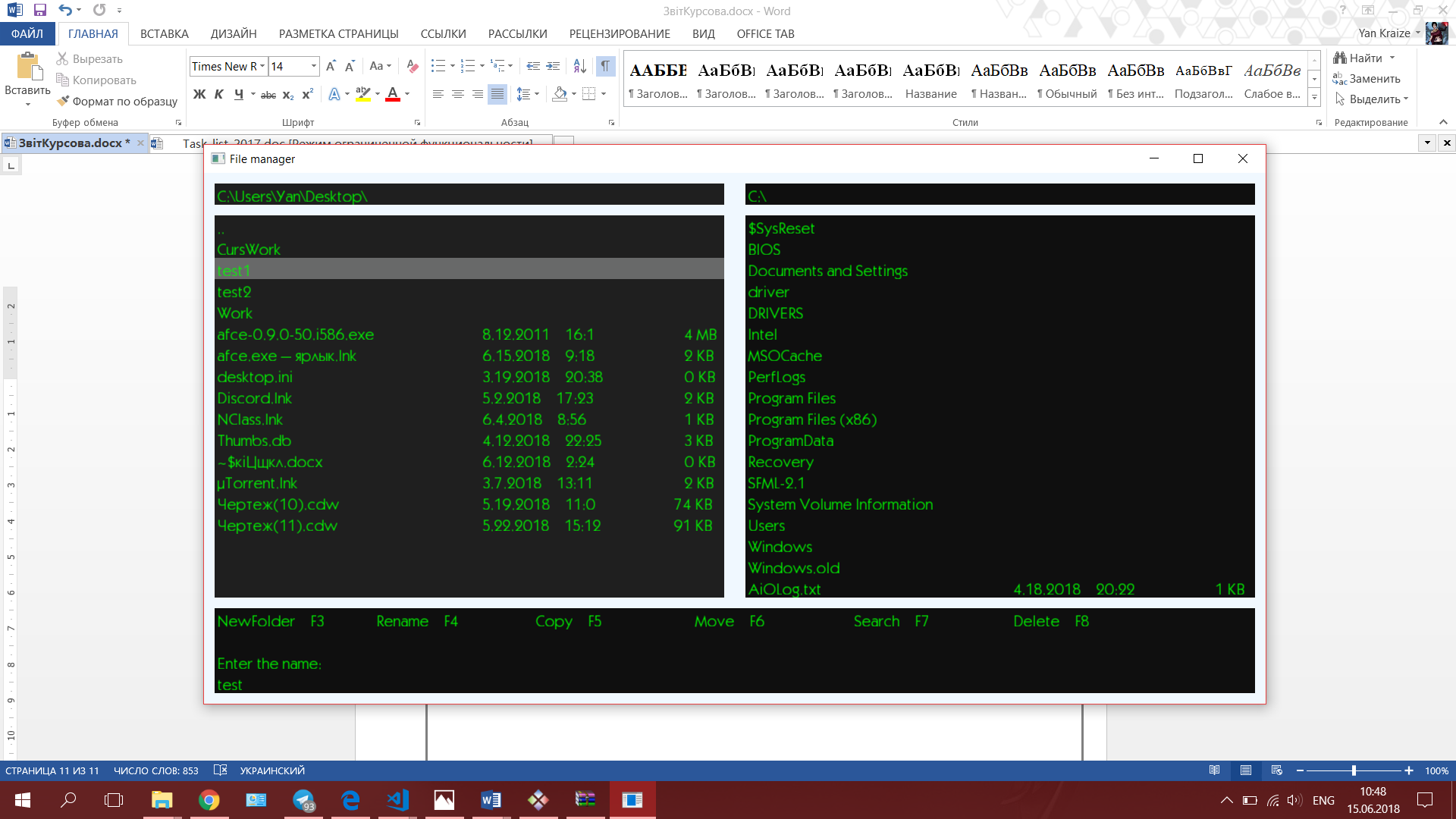
У цій роботі мною було створено повноцінній працюючий програмний продукт. Я покращив свої знання С++ і SFML, познайомився з бібліотекою Win API. В пояснювальній записці надано детальний опис програми, керівництва для користувача та розробника. Дана програма працює коректно, тому можна зробити висновок, що поставлене завдання виконано.

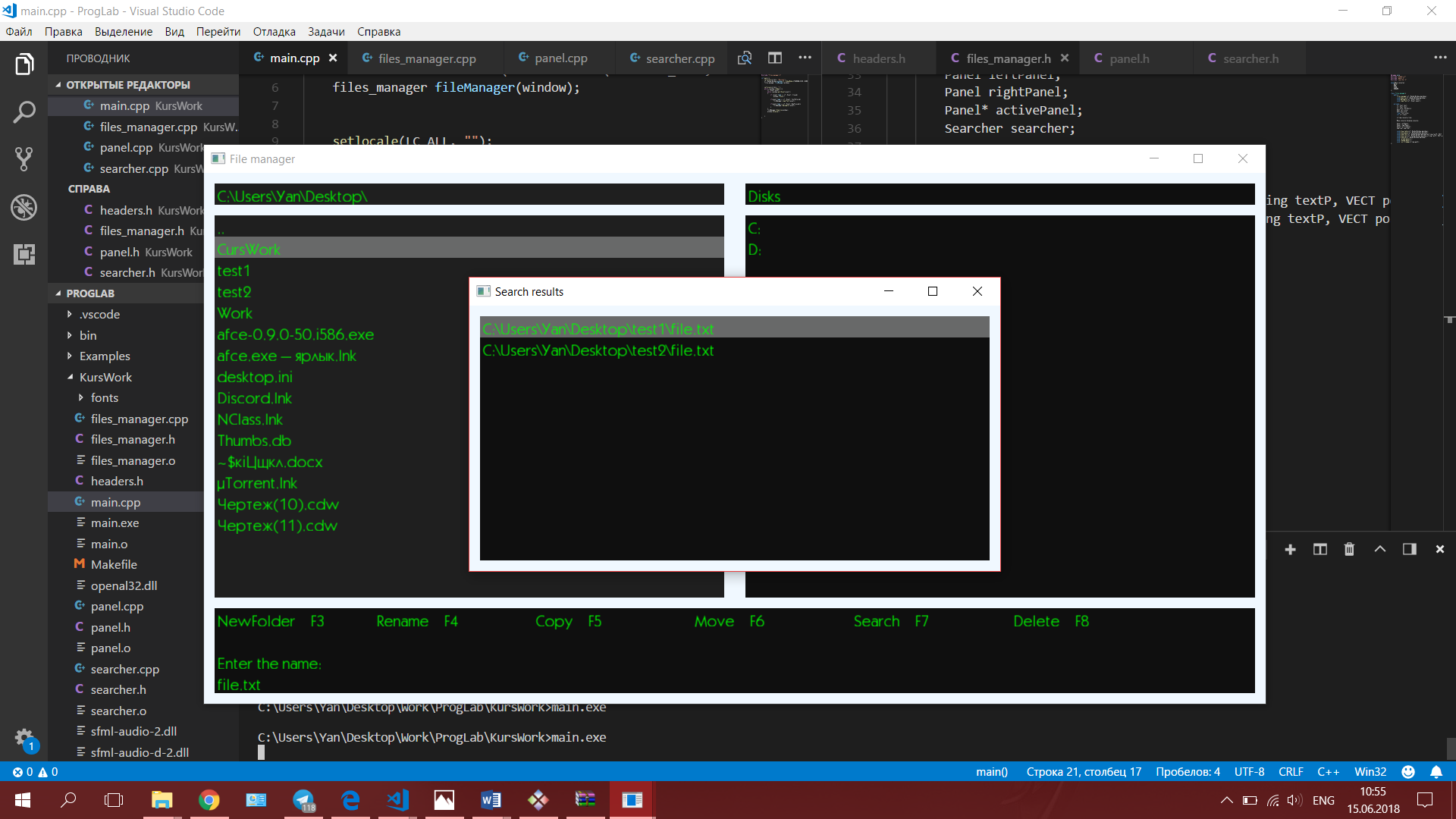
### Джерела

1. <https://msdn.microsoft.com/en-us/library/windows/desktop/aa363946(v=vs.85).aspx>
2. <https://www.sfml-dev.org/>
3. <https://ru.stackoverflow.com/>

# Додаток 1







# Додаток 2

main.cpp

#include "files\_manager.h"

int main(){

FreeConsole();

setlocale(LC\_ALL, "Russian");

sf::RenderWindow window(sf::VideoMode(2\*WINDOW\_SIZE, WINDOW\_SIZE), "File manager");

files\_manager fileManager(window);

while (window.isOpen()){

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

if(event.type == sf::Event::TextEntered)

fileManager.inputText(event);

if(event.type == sf::Event::KeyPressed){

fileManager.Navigator(event);

}

}

fileManager.Explore(window);

window.display();

}

}

file\_manager.cpp

#include "files\_manager.h"

files\_manager::files\_manager(sf::RenderWindow &window):cell\_size(VECT(CELL\_SIZE, CELL\_SIZE)),

win\_size(VECT(window.getSize().x/cell\_size.x, window.getSize().y/cell\_size.y)), font\_size(cell\_size.y\*1.5),

str\_count((win\_size.y - 14) / 2), display\_console(Menu\_console::NO),searcher(){

if(!font.loadFromFile("fonts/Azbuka.ttf"))

cout<<"Error font loading"<<endl;

bordColor = sf::Color(240, 248, 255);

activePanel = &leftPanel;

text.setCharacterSize(font\_size);

text.setFont(font);

}

void files\_manager::Explore(sf::RenderWindow &window){

sf::RectangleShape bg(VECT(win\_size.x\*cell\_size.x, win\_size.y\*cell\_size.y));

bg.setFillColor(sf::Color(15,15,15));

window.draw(bg);

draw\_table(window);

draw\_hb(window);

draw\_panel(window);

if(display\_console){

draw\_text(window,"Enter the name:", VECT(1, win\_size.y - 5));

draw\_text(window, console\_line, VECT(1, win\_size.y - 3));

}

}

void files\_manager::draw\_panel(sf::RenderWindow &window){

sf::RectangleShape rect(cell\_size);

rect.setFillColor(sf::Color(31,31,31));

if(activePanel==&leftPanel){

for(int x = 1; x < win\_size.x/2 - 1; x++){

rect.setPosition(x \* cell\_size.x, cell\_size.y);

window.draw(rect);

rect.setPosition(x \* cell\_size.x, 2\*cell\_size.y);

window.draw(rect);

}

for(int x = 1; x < win\_size.x/2 - 1; x++){

for(int y = 4; y < win\_size.y - 10; y++){

rect.setPosition(x \* cell\_size.x, y \* cell\_size.y);

window.draw(rect);

}

}

}

else{

for(int x = win\_size.x/2 + 1; x < win\_size.x - 1; x++){

rect.setPosition(x \* cell\_size.x, cell\_size.y);

window.draw(rect);

rect.setPosition(x \* cell\_size.x, 2\*cell\_size.y);

window.draw(rect);

}

for(int x = win\_size.x/2 + 1; x < win\_size.x - 1; x++){

for(int y = 4; y < win\_size.y - 10; y++){

rect.setPosition(x \* cell\_size.x, y \* cell\_size.y);

window.draw(rect);

}

}

}

draw\_text(window, leftPanel.current\_directory, VECT(1,1));

draw\_text(window, rightPanel.current\_directory, VECT(win\_size.x/2 + 1,1));

VECT line(1, 4);

sf::RectangleShape hilight(VECT((win\_size.x/2 - 2)\* cell\_size.x, 2\* cell\_size.y));

hilight.setFillColor(sf::Color(104,105,105));

//draw leftPanel.content

for(int i = leftPanel.start\_index;

i<leftPanel.start\_index+(str\_count<leftPanel.content.size()?str\_count:leftPanel.content.size());

i++){

if(leftPanel.cursor == i && activePanel==&leftPanel){

hilight.setPosition(line.x\*cell\_size.x, line.y\*cell\_size.y);

window.draw(hilight);

}

draw\_text(window, leftPanel.content[i].name, line);

draw\_text(window, leftPanel.content[i].time, VECT(line.x + 25, line.y));

draw\_text(window, leftPanel.content[i].size, VECT(win\_size.x/2 - 1 - leftPanel.content[i].size.size(), line.y));

line.y+=2;

}

line = VECT(win\_size.x/2 + 1, 4);

//draw rightPanel.content

for(int i = rightPanel.start\_index;

i<rightPanel.start\_index+(str\_count<rightPanel.content.size()?str\_count:rightPanel.content.size());

i++){

if(rightPanel.cursor == i && activePanel==&rightPanel){

hilight.setPosition(line.x\*cell\_size.x, line.y\*cell\_size.y);

window.draw(hilight);

}

draw\_text(window, rightPanel.content[i].name, line);

draw\_text(window, rightPanel.content[i].time, VECT(line.x + 25, line.y));

draw\_text(window, rightPanel.content[i].size, VECT(win\_size.x - 1 - rightPanel.content[i].size.size(), line.y));

line.y+=2;

}

}

void files\_manager::draw\_hb(sf::RenderWindow &window){

int tab=15;

VECT pos(1, win\_size.y - 9);

draw\_text(window, "NewFolder\tF3\0", pos);

pos.x += tab;

draw\_text(window, "Rename\tF4\0", pos);

pos.x += tab;

draw\_text(window, "Copy\tF5\0", pos);

pos.x += tab;

draw\_text(window, "Move\tF6\0", pos);

pos.x += tab;

draw\_text(window, "Search\tF7\0", pos);

pos.x += tab;

draw\_text(window, "Delete\tF8\0", pos);

}

void files\_manager::draw\_table(sf::RenderWindow &window){

sf::RectangleShape rect(cell\_size);

rect.setFillColor(bordColor);

for(int x = 0; x < win\_size.x; x++){ //horisontal borders

rect.setPosition(x \* cell\_size.x, 0);

window.draw(rect);

rect.setPosition(x \* cell\_size.x, (win\_size.y - 1)\* cell\_size.y);

window.draw(rect);

}

for(int y = 0; y < win\_size.y; y++){ //vertical borders

rect.setPosition(0, y\*cell\_size.y);

window.draw(rect);

rect.setPosition((win\_size.x - 1)\* cell\_size.x, y \* cell\_size.y);

window.draw(rect);

}

for(int x = 0; x < win\_size.x; x++){ //cur\_dir line and help\_box separator

rect.setPosition(x\* cell\_size.x, 3\*cell\_size.y);

window.draw(rect);

rect.setPosition(x\* cell\_size.x, (win\_size.y - 10)\* cell\_size.y);

window.draw(rect);

}

for(int y = 0; y < win\_size.y - 10; y++){ //panel separator

rect.setPosition((win\_size.x/2)\* cell\_size.x, y\*cell\_size.y);

window.draw(rect);

rect.setPosition((win\_size.x/2 - 1)\* cell\_size.x, y \* cell\_size.y);

window.draw(rect);

}

}

void files\_manager::draw\_text(sf::RenderWindow &window, string textP, VECT position){

text.setString(textP);

text.setFillColor(sf::Color::Green);

text.setPosition(cell\_size.x\*(position.x + 0.25), cell\_size.y\*(position.y + 0.25));

window.draw(text);

}

//for russian languge use L"Сообщение "

void files\_manager::draw\_text(sf::RenderWindow &window, wstring textP, VECT position){

text.setString(textP);

text.setFillColor(sf::Color::Green);

text.setPosition(cell\_size.x\*(position.x + 0.25), cell\_size.y\*(position.y + 0.25));

window.draw(text);

}

void files\_manager::Navigator(sf::Event event){

if(!display\_console){

switch(event.key.code){

case sf::Keyboard::Escape:

if(display\_console != Menu\_console::NO){

display\_console = Menu\_console::NO;

console\_line.erase();

}

else

activePanel->logical\_drives();

break;

case sf::Keyboard::Up:

if(!(activePanel->cursor-1<0))

activePanel->cursor--;

if(activePanel->cursor < activePanel->start\_index)

activePanel->start\_index=activePanel->cursor;

break;

case sf::Keyboard::Down:

if(activePanel->cursor+1<activePanel->content.size())

activePanel->cursor++;

if(activePanel->cursor > activePanel->start\_index + str\_count-1)

activePanel->start\_index++;

break;

case sf::Keyboard::Left:

change\_panel();

break;

case sf::Keyboard::Right:

change\_panel();

break;

case sf::Keyboard::Return:

if(activePanel->is\_current\_Directory())

activePanel->change\_dir(activePanel->folders\_to\_display[activePanel->cursor].name);

else

run\_fileApp(activePanel->current\_directory+'\\'+ activePanel->files\_to\_display[activePanel->cursor-activePanel->folders\_to\_display.size()].name);

break;

case sf::Keyboard::F3:

display\_console = Menu\_console::NEW;

break;

case sf::Keyboard::F4:

display\_console = Menu\_console::RENAME;

return;

case sf::Keyboard::F5:{

string from\_dir = activePanel->current\_directory;

string name;

if(activePanel->is\_current\_Directory())

name = activePanel->folders\_to\_display[activePanel->cursor].name;

else

name = activePanel->files\_to\_display[activePanel->cursor - activePanel->folders\_to\_display.size()].name;

change\_panel();

activePanel->copy(from\_dir, name);

reload\_panels();

break;}

case sf::Keyboard::F6:{

string from\_dir = activePanel->current\_directory;

string name;

if(activePanel->is\_current\_Directory())

name = activePanel->folders\_to\_display[activePanel->cursor].name;

else

name = activePanel->files\_to\_display[activePanel->cursor - activePanel->folders\_to\_display.size()].name;

change\_panel();

activePanel->move(from\_dir, name);

reload\_panels();

break;}

case sf::Keyboard::F7:

display\_console = Menu\_console::SCAN;

break;

case sf::Keyboard::F8:

activePanel->del();

break;;

}

}

else{if(event.key.code == sf::Keyboard::Escape){

display\_console = Menu\_console::NO;

console\_line.erase();

}

else if(event.key.code == sf::Keyboard::Return){

switch(display\_console) {

case Menu\_console::NEW:

activePanel->create\_dir(console\_line);

reload\_panels();

break;

case Menu\_console::RENAME:

activePanel->rename(console\_line);

reload\_panels();

break;

case Menu\_console::SCAN:

vector<string> found;

activePanel->search(console\_line, found);

if(!found.empty()){

searcher.found = found;

searcher.search\_display();

if(!searcher.path\_to\_go.empty()){

if(searcher.path\_to\_go.back() == '\\'){

activePanel->logical\_drives();

std::size\_t foundp = searcher.path\_to\_go.find\_last\_of("\\");

activePanel->change\_dir(searcher.path\_to\_go.substr(0, foundp));

}

else{

run\_fileApp(searcher.path\_to\_go);

}

}

}

else

MessageBox( NULL,TEXT("No results"),TEXT("Search"),MB\_ICONINFORMATION |MB\_OK);

searcher.path\_to\_go.erase();

break;

}

display\_console = Menu\_console::NO;

console\_line.erase();

}

}

}

void files\_manager::reload\_panels(){

string temp = activePanel->current\_directory;

if(temp != "Disks"){

activePanel->logical\_drives();

std::size\_t foundp = temp.find\_last\_of('\\');

activePanel->change\_dir(temp.substr(0, foundp));

}

change\_panel();

temp = activePanel->current\_directory;

if(temp != "Disks"){

activePanel->logical\_drives();

std::size\_t foundp = temp.find\_last\_of('\\');

activePanel->change\_dir(temp.substr(0, foundp));

}

change\_panel();

}

void files\_manager::change\_panel(){

if(activePanel == &leftPanel)

activePanel = &rightPanel;

else

activePanel = &leftPanel;

}

void files\_manager::inputText(sf::Event event){

// NO=0, NEW=1, SCAN=2, RENAME=3

if(display\_console){

if(event.key.code =='\b' && !console\_line.empty()){

console\_line.pop\_back();

}

else{

char buffer[2];

wchar\_t strW[2];

strW[0]=static\_cast<wchar\_t>(event.text.unicode);

wcstombs (buffer, strW, sizeof(buffer));

if(strpbrk(buffer,"<>:\"\\|/?\*")==NULL)

console\_line.push\_back(buffer[0]);

else{

MessageBox( NULL,

TEXT("File and directory names can not contain the following characters \n\t\" <>:\"\\|/?\* \" !"),

TEXT("Input text"),

MB\_ICONWARNING |MB\_OK);

}

}

}

}

void files\_manager::run\_fileApp(string path){

HINSTANCE result = ShellExecute(NULL,NULL,path.c\_str(),NULL,NULL,SW\_RESTORE);

long long sig1 = reinterpret\_cast<long long> (result);

int sig = static\_cast<int>(sig1);

if(sig < 32){

MessageBox( NULL,TEXT("Ooops! There is some problem!"),TEXT("Open file"),MB\_ICONWARNING | MB\_OK);

//https://msdn.microsoft.com/en-us/library/windows/desktop/bb762153(v=vs.85).aspx

//all return code -which problem

}

}

panel.cpp

#include "panel.h"

Panel::Panel(){

logical\_drives();

}

void Panel::logical\_drives(){

current\_directory = "Disks";

cursor = 0;

start\_index = 0;

folders\_to\_display.clear();

files\_to\_display.clear();

content.clear();

DWORD dir = GetLogicalDrives();

string temp;

for(int i = 0; i < (sizeof(DWORD)\*8); i++){

if(dir & 1<<i){

folders\_to\_display.push\_back(folder());

temp = static\_cast<char>(i + 65);

folders\_to\_display.back().name = (temp + ":" + "\0");

content.push\_back(container());

content.back().name = folders\_to\_display.back().name;

content.back().size = "";

content.back().time = "";

}

}

}

void Panel::create\_dir(string name){

\_SECURITY\_ATTRIBUTES security;

security.nLength = sizeof(SECURITY\_ATTRIBUTES);

security.lpSecurityDescriptor = NULL;

security.bInheritHandle = TRUE;

string path = current\_directory + name;

if (path.length() > (MAX\_PATH - 3)){

cout<<"Directory path is too long.\n";

return;

}

if(current\_directory=="Disks"){

MessageBox(NULL, "You can't create folders here!",TEXT("Create directory"), MB\_ICONWARNING|MB\_OK);

return;

}

if(!CreateDirectory(path.c\_str(), &security)){

MessageBox(NULL, "Not enough rihgts!",TEXT("Create directory") ,MB\_ICONINFORMATION|MB\_OK);

return;

}

change\_dir(name);

}

void Panel::change\_dir(string path){

if (path.length() > (MAX\_PATH - 3)){

MessageBox(NULL, "Directory path is too long!",TEXT("Change directory"), MB\_ICONWARNING|MB\_OK);

return;

}

string dir = current\_directory=="Disks"?"":current\_directory;

if(path != ".."){

path += "\\";

dir+=path;

}

else{

std::size\_t found = dir.find\_last\_of("\\");

dir = dir.substr(0, found);

found = dir.find\_last\_of("\\");

dir = dir.substr(0, found + 1);

}

WIN32\_FIND\_DATA ffd;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

dir+="\*";

hFind = FindFirstFileA(dir.c\_str(), &ffd);

if (INVALID\_HANDLE\_VALUE == hFind){

MessageBox(NULL, "Ooops! Directory has some problem",TEXT("Change directory"), MB\_ICONWARNING|MB\_OK);

return ;

}

//меняем значение current\_directory после успешного FindFirstFileA(...)

current\_directory=dir.substr(0,dir.size()-1);

cursor=0;

start\_index=0;

folders\_to\_display.clear();

files\_to\_display.clear();

content.clear();

while (FindNextFile(hFind, &ffd) != 0){

if (ffd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY){

folders\_to\_display.push\_back(folder());

folders\_to\_display.back().name = ffd.cFileName;

folders\_to\_display.back().time.LowPart = ffd.ftCreationTime.dwLowDateTime;

folders\_to\_display.back().time.HighPart = ffd.ftCreationTime.dwHighDateTime;

}

else{

files\_to\_display.push\_back(file());

files\_to\_display.back().name = ffd.cFileName;

files\_to\_display.back().size.LowPart = ffd.nFileSizeLow;

files\_to\_display.back().size.HighPart = ffd.nFileSizeHigh;

files\_to\_display.back().time = ffd.ftLastWriteTime;

}

}

FindClose(hFind);

get\_content();

}

void Panel::del(){

if(is\_current\_Directory()){

if(current\_directory=="Disks")

return;

else

delete\_dir();

}

else

delete\_file();

}

void Panel::delete\_file(){

int index\_file = this->cursor - this->folders\_to\_display.size();

string file\_name = this->files\_to\_display[index\_file].name;

string message("Delete "+file\_name +"\nAre you shure ? ");

int result = MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete file"),MB\_ICONQUESTION |MB\_YESNO);

if( result==IDNO)

return;

else if(result==IDYES){

BOOL fSuccess = FALSE;

string full\_path (current\_directory+'\\'+file\_name);

fSuccess= DeleteFile(TEXT(full\_path.c\_str()));

if (!fSuccess){

message.assign("Can not to delete "+file\_name);

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete file"),MB\_ICONWARNING |MB\_OK);

}

else{

message.assign(file\_name+" file was deleted.");

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete file"),MB\_ICONINFORMATION |MB\_OK);

files\_to\_display.erase(files\_to\_display.begin()+index\_file);

content.erase(content.begin()+cursor);

cursor--;

}

}

}

void Panel::delete\_dir(){

string dir\_name = this->folders\_to\_display[cursor].name;

string message("Delete "+dir\_name +"\nAre you shure ? ");

int result = MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete directory"),MB\_ICONQUESTION |MB\_YESNO);

if(result == IDNO)

return;

else if(result == IDYES){

dir\_name.resize(dir\_name.length()+2);

string dir\_path(current\_directory+'\\'+dir\_name);

SHFILEOPSTRUCT shfo = {

NULL,

FO\_DELETE,

dir\_path.c\_str(),

NULL,

FOF\_SILENT | FOF\_NOERRORUI | FOF\_NOCONFIRMATION,

FALSE,

NULL,

NULL };

bool fSuccess=(SHFileOperation(&shfo) == 0);

if (!fSuccess){

message.assign("Can not to delete "+dir\_name);

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete directory"),MB\_ICONWARNING |MB\_OK);

}

else{

message.assign(dir\_name+" directory was deleted.");

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Delete directory"),MB\_ICONINFORMATION |MB\_OK);

folders\_to\_display.erase(folders\_to\_display.begin()+cursor);

content.erase(content.begin()+cursor);

cursor--;

}

}

}

void Panel::copy(string fromPath, string name){

if(current\_directory == "Disks" || fromPath == "Disks"){

string message("You can't copy logica driver or to logical drivers!");

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Copy"),MB\_ICONWARNING |MB\_OK);

return;

}

name.resize(name.length()+2);

string dir\_path\_from(fromPath+'\\'+name);

string dir\_path\_to(current\_directory+'\\'+name);

SHFILEOPSTRUCT shfo = {

NULL,

FO\_COPY,

dir\_path\_from.c\_str(),

dir\_path\_to.c\_str(),

FOF\_SILENT | FOF\_NOERRORUI | FOF\_NOCONFIRMATION,

FALSE,

NULL,

NULL };

if(!SHFileOperation(&shfo)==0){

string message("Can not copy "+ name);

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Copy"),MB\_ICONWARNING |MB\_OK);

}

}

void Panel::move(string fromPath, string name){

if(current\_directory == "Disks" || fromPath == "Disks"){

string message("You can't move logical driver or to logical drivers!");

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Move"),MB\_ICONWARNING |MB\_OK);

return;

}

name.resize(name.length()+2);

string dir\_path\_from(fromPath+'\\'+name);

string dir\_path\_to(current\_directory+'\\'+name);

SHFILEOPSTRUCT shfo = {

NULL,

FO\_MOVE,

dir\_path\_from.c\_str(),

dir\_path\_to.c\_str(),

FOF\_SILENT | FOF\_NOERRORUI | FOF\_NOCONFIRMATION,

FALSE,

NULL,

NULL };

if( !SHFileOperation(&shfo)==0){

string message("Can not move "+ name);

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Move"),MB\_ICONWARNING |MB\_OK);

}

}

void Panel::rename(string name){

if(current\_directory == "Disks"){

string message("You can't rename logica drivers!");

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Rename"),MB\_ICONWARNING |MB\_OK);

}

string dir\_path\_from;

string dir\_path\_to;

///

if(is\_current\_Directory()){

dir\_path\_from = current\_directory+'\\'+folders\_to\_display[cursor].name;

dir\_path\_to = current\_directory+'\\'+name;

}

else{

dir\_path\_from = current\_directory+'\\'+files\_to\_display[cursor - folders\_to\_display.size()].name;

if(name.find\_last\_of('.') != string::npos){

string message("Do you want to change files extention?");

int result = MessageBox( NULL,TEXT(message.c\_str()),TEXT("Rename"),MB\_ICONQUESTION |MB\_YESNO);

if(result == IDYES)

dir\_path\_to = current\_directory+'\\'+name;

else if(result == IDNO){

std::size\_t foundp = files\_to\_display[cursor - folders\_to\_display.size()].name.find\_last\_of('.');

string fname = name + files\_to\_display[cursor - folders\_to\_display.size()].name.substr(foundp);

dir\_path\_to = current\_directory+'\\'+fname;

}

}

else{

std::size\_t foundp = files\_to\_display[cursor - folders\_to\_display.size()].name.find\_last\_of('.');

string fname = name + files\_to\_display[cursor - folders\_to\_display.size()].name.substr(foundp);

dir\_path\_to = current\_directory+'\\'+fname;

}

}

dir\_path\_from.resize(dir\_path\_from.length()+2);

dir\_path\_to.resize(dir\_path\_to.length()+2);

SHFILEOPSTRUCT shfo = {

NULL,

FO\_RENAME,

dir\_path\_from.c\_str(),

dir\_path\_to.c\_str(),

FOF\_SILENT | FOF\_NOERRORUI | FOF\_NOCONFIRMATION,

FALSE,

NULL,

NULL };

int a = SHFileOperation(&shfo);

if( a != 0 ){

string message("Can not rename "+ name);

MessageBox( NULL,TEXT(message.c\_str()),TEXT("Rename"),MB\_ICONWARNING |MB\_OK);

cout<<hex<<a<<endl;

}

}

void Panel::search(string name, vector<string>& found){

queue<string> subfolders;

subfolders.push(current\_directory);

while(!subfolders.empty()){ // пока очередь посещения клеток непустая

string cur = subfolders.front();

subfolders.pop();

search\_in\_folder(name, cur, found, subfolders);

}

return;

}

void Panel::search\_in\_folder(string name, string path, vector<string>& found, queue<string>& subfolders){

if (path.length() > (MAX\_PATH - 3)){

cout<<"Directory path is too long.\n";

return;

}

string dir = path;

WIN32\_FIND\_DATA ffd;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

dir+="\*";

hFind = FindFirstFileA(dir.c\_str(), &ffd);

if (INVALID\_HANDLE\_VALUE == hFind){

cout<<"3 "<<current\_directory<<'\t'<<dir<<endl;

return ;

}

FindNextFile(hFind, &ffd);

while (FindNextFile(hFind, &ffd) != 0){

if (ffd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY){

if(ffd.cFileName == name)

found.push\_back(path + ffd.cFileName + '\\');

subfolders.push(path + ffd.cFileName + '\\');

}

else{

if(ffd.cFileName == name)

found.push\_back(path + ffd.cFileName);

}

}

FindClose(hFind);

}

void Panel::get\_content(){

for(int i = 0; i < folders\_to\_display.size(); i++){

content.push\_back(container());

content.back().name = folders\_to\_display[i].name;

content.back().size = "";

content.back().time = "";

}

for(int i = 0; i < files\_to\_display.size(); i++){

content.push\_back(container());

string line = files\_to\_display[i].name;

while(line.size() > 25){

for(int j = 0; j < 3; j++)

line.pop\_back();

line += "..";

}

content.back().name = line;

line.erase();

SYSTEMTIME stUTC, stLocal;

FileTimeToSystemTime(&files\_to\_display[i].time, &stUTC);

SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);

line += to\_string(stLocal.wMonth);

line += '.';

line += to\_string(stLocal.wDay);

line += '.';

line += to\_string(stLocal.wYear);

line += '\t';

line += to\_string(stLocal.wHour);

line += ':';

line += to\_string(stLocal.wMinute);

line += '\t';

content.back().time = line;

if(files\_to\_display[i].size.QuadPart < pow(1024,2))

line = to\_string(INT(files\_to\_display[i].size.QuadPart / 1024)) + " KB";

else if(files\_to\_display[i].size.QuadPart < pow(1024,3))

line = to\_string(INT(files\_to\_display[i].size.QuadPart / pow(1024,2))) + " MB";

else if(files\_to\_display[i].size.QuadPart < pow(1024,4))

line = to\_string(INT(files\_to\_display[i].size.QuadPart / pow(1024,3))) + " GB";

content.back().size = line;

}

}

searcher.cpp

#include "searcher.h"

Searcher::Searcher():cell\_size(VECT(CELL\_SIZE, CELL\_SIZE)),

font\_size(cell\_size.y\*1.5){

if(!font.loadFromFile("fonts/Azbuka.ttf"))

cout<<"Error font loading"<<endl;

s\_cursor = 0;

start\_index = 0;

}

void Searcher::search\_display(){

search\_results.create(sf::VideoMode(WINDOW\_SIZE, 0.5\*WINDOW\_SIZE), "Search results");

search\_win\_size = VECT(search\_results.getSize().x/cell\_size.x, search\_results.getSize().y/cell\_size.y);

str\_count = (search\_win\_size.y - 14) / 2;

while (search\_results.isOpen()){

sf::Event event;

while (search\_results.pollEvent(event))

{

if (event.type == sf::Event::Closed){

path\_to\_go.erase();// no action

search\_results.close();

}

if(event.type == sf::Event::KeyPressed){

switch(event.key.code){

case sf::Keyboard::Up:

if(!(s\_cursor-1<0))

s\_cursor--;

if(s\_cursor < start\_index){

start\_index = s\_cursor;

}

break;

case sf::Keyboard::Down:

if(s\_cursor+1<found.size())

s\_cursor++;

if(s\_cursor > start\_index + str\_count-1)

start\_index++;

break;

case sf::Keyboard::Return:{

path\_to\_go = found[s\_cursor];

search\_results.close();

break;}

case sf::Keyboard::Escape:

path\_to\_go.erase();// no action

search\_results.close();

break;

}

}

}

search\_draw();

search\_results.display();

}

}

void Searcher::search\_draw(){

sf::RectangleShape bg(VECT(search\_win\_size.x\*cell\_size.x, search\_win\_size.y\*cell\_size.y));

bg.setFillColor(sf::Color(15,15,15));

search\_results.draw(bg);

sf::RectangleShape rect(cell\_size);

rect.setFillColor(sf::Color(240, 248, 255));

for(int x = 0; x < search\_win\_size.x; x++){ //horisontal borders

rect.setPosition(x \* cell\_size.x, 0);

search\_results.draw(rect);

rect.setPosition(x \* cell\_size.x, (search\_win\_size.y - 1)\* cell\_size.y);

search\_results.draw(rect);

}

for(int y = 0; y < search\_win\_size.y; y++){ //vertical borders

rect.setPosition(0, y\*cell\_size.y);

search\_results.draw(rect);

rect.setPosition((search\_win\_size.x - 1)\* cell\_size.x, y \* cell\_size.y);

search\_results.draw(rect);

}

sf::RectangleShape hilight(VECT((search\_win\_size.x - 2)\* cell\_size.x, 2\* cell\_size.y));

hilight.setFillColor(sf::Color(104,105,105));

VECT line(1, 1);

for(int i = start\_index; i < start\_index+(str\_count<found.size()?str\_count:found.size()); i++){

if(s\_cursor == i){

hilight.setPosition(line.x\*cell\_size.x, line.y\*cell\_size.y);

search\_results.draw(hilight);

}

draw\_text(search\_results, found[i], line);

line.y+=2;

}

}

void Searcher::draw\_text(sf::RenderWindow &window, string textP, VECT position){

sf::Text text(textP, font);

text.setCharacterSize(font\_size);

text.setFillColor(sf::Color::Green);

text.setPosition(cell\_size.x\*(position.x + 0.25), cell\_size.y\*(position.y + 0.25));

window.draw(text);

}

# Додаток 3

