НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

до лабораторної роботи №3

з дисципліни «Операційні системи»

на тему

«Повiдомлення та черги повiдомлень в ОС Windows»

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-51 | Громова В. В. |
| Лук’яненко А. М. |  |

Київ — 2018

ЗМІСТ

[ВСТУП 3](#_Toc509180983)

[1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc509180984)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 5](#_Toc509180985)

[2.1 Вiконнi процедури 5](#_Toc509180986)

[2.2 Повiдомлення Windows 5](#_Toc509180987)

[2.3 Маршрутизацiя повiдомлень 6](#_Toc509180988)

[2.4 Цикл опрацювання повiдомлень 7](#_Toc509180989)

[3 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМИ 8](#_Toc509180990)

[4 РЕЗУЛЬТАТИ ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ 15](#_Toc509180991)

[ВИСНОВКИ 17](#_Toc509180992)

[ПОСИЛАННЯ НА ЛIТЕРАТУРНI ДЖЕРЕЛА 18](#_Toc509180993)

[Додаток А Текст програми 19](#_Toc509180994)

# ВСТУП

На вiдмiну вiд аплiкацiй, написаних для операцiйної системи (ОС) MS-DOS, аплiкацiї, написанi для ОС Microsoft Windows, керованi за допомогою подiй. Вони не здiйснюють явнi виклики функцiй (наприклад, виклики бiблiотеки мови програмування C) для отримання уведення вiд користувача. Натомiсть, вони ждуть, щоб ОС передала їм повiдомлення у разi, якщо таке уведення матиме мiсце.

ОС Windows передає все уведення, призначене для деякої аплiкацiї, рiзним вiкнам цiєї аплiкацiї. Кожне вікно має функцію, яку називають вiконною процедурою, яку Windows викликає, щоразу для вiкна з’являється уведення. Вiконна процедура опрацьовує уведення та повертає керування ОС.

У данiй роботi розглянуті засоби роботи з повiдомленнями в ОС Windows версiї 2000 та вище.

# 1 ПОСТАНОВКА ЗАДАЧІ

У данiй лабораторнiй роботi потрiбно ознайомитися з концепцiєю повiдомлень в ОС Windows, отримати навички використання засобiв Windows API для роботи з повiдомленнями для поєднання рiзних компонентiв аплiкацiї або рiзних аплiкацiй мiж собою.

У рамках виконання лабораторної роботи потрiбно:

1. ознайомитися з теоретичними вiдомостями;
2. написати будь-якою мовою програмування програму, яка повинна:

* використовувати стандартнi повiдомлення Windows для уведення з клавiатури та за допомогою мишi, обмiну даними з елементами керування користувацького iнтерфейсу, реагування на команди меню, керування вiкнами;
* використовувати користувацькi повiдомлення для обмiну даними мiж компонентами аплiкацiї;
* використовувати тiльки засоби Windows API i не повинна використовувати стандартнi процедури MFC тощо;

1. вiдлагодити програму в ОС Windows версiї 2000 та вище;
2. пiдготувати звiт iз лабораторної роботи вiдповiдно до вимог.

Програма повинна містити iндивiдуальні функцiональні можливості.

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

## 2.1 Вiконнi процедури

Кожне вiкно в ОС Windows — об’єкт певного вiконного класу (window class). Вiконний клас визначає вiконну процедуру (window procedure), яку конкретне вiкно використовує для опрацювання повiдомлень. Вiконна процедура — це функцiя, котра опрацьовує всi повiдомлення, надiсланi вiкну певного класу.

Як правило, аплiкацiя реєструє принаймнi один новий вiконний клас та асоцiйовану з ним вiконну процедуру. Пiсля реєстрацiї класу, аплiкацiя може створювати багато вiкон цього класу, кожне з яких використовуватиме одну й ту саму вiконну процедуру.

Оскiльки вiконна процедура спiльна для всiх вiкон одного класу, вона може отримувати повiдомлення, призначенi для рiзних вiкон. Для iдентифiкацiї конкретного вiкна, якому адресовано повiдомлення, вiконна процедура може аназiлувати дескриптор вiкна, який передають разом iз повiдомленням.

## 2.2 Повiдомлення Windows

ОС Windows передає вiконнiй процедурi уведення у формi повiдомлень (messages). Повiдомлення можуть генерувати як Windows, так i аплiкацiї:

* Windows генерує повiдомлення пiд час кожної подiї уведення та у вiдповiдь на змiни в системi, якi викликає аплiкацiя;
* аплiкацiя генерує повiдомлення, щоб змусити власнi вiкна виконати певнi задачi або щоб зв’язатися з вiкнами iнших аплiкацiй.

Windows надсилає вiконнiй процедурi повiдомлення з чотирма параметрами:

* дескриптор вiкна (window handle). Дескриптор — це беззнакове 32-бiтове числове значення, яке ОС використовує для iдентифiкацiї об’єктiв у цiй таблицi;
* iдентифiкатор повiдомлення (message identifier) — iменована 32-бiтова константа, яка iдентифiкує призначення повiдомлення. Вiконна процедура використовує це значення, щоб визначити, як його варто опрацьовувати;
* два 32-розряднi значення — параметри повiдомлення (message parameters). Цi параметри визначають данi (або їхнє розташування), якi вiконна процедура може використовувати пiд час опрацювання повiдомлення.

## 2.3 Маршрутизацiя повiдомлень

ОС Windows використовує два методи надсилання повiдомлень вiконнiй процедурi:

* надсилання (posting) повiдомлень у чергу повiдомлень (message queue) — системний об’єкт пам’ятi, який тимчасово зберiгає повiдомлення. Повiдомлення, поставленi в чергу повiдомлень (черговi повiдомлення, queued messages), — це результати уведення користувача за допомогою мишi чи клавiатури, а також повiдомлення таймера (WM\_TIMER), перемалювання (WM\_PAINT) та виходу (WM\_QUIT);
* надсилання (sending) повiдомлень вiконнiй процедурi безпосередньо. Такi повiдомлення називають позачерговими (non-queued messages).

## 2.4 Цикл опрацювання повiдомлень

Аплiкацiя повинна вибирати повiдомлення з черг своїх потокiв та опрацьовувати їх. Однопотокова аплiкацiя зазвичай використовує для цього цикл опрацювання повiдомлень у своїй функцiї WinMain (WinMain — це точка входу в аплiкацiю Windows). Багатопотокова аплiкацiя може мiстити такий цикл у кожному своєму потоцi, який створює вiкно.

Головний потiк аплiкацiї починає цикл опрацювання повiдомлень пiсля iнiцiалiзацiї аплiкацiї та створення щонайменше одного вiкна. На кожнiй iтерацiї циклу вибирають одне повiдомлення з черги та направляють його потрiбнiй вiконнiй процедурi. Цикл завершує свою роботу, коли GetMessage вибирає з черги повiдомлення WM\_QUIT.

# 3 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМИ

Для того, щоб код програми відповідав нормам, він має починатися із закоментованого абзацу, що коротко описує його – це ознака норми та гарного тону. В ньому зазвичай міститься назва програми та ім’я автору. У більш розширених версіях є логотип компанії, час створення та опис програми.

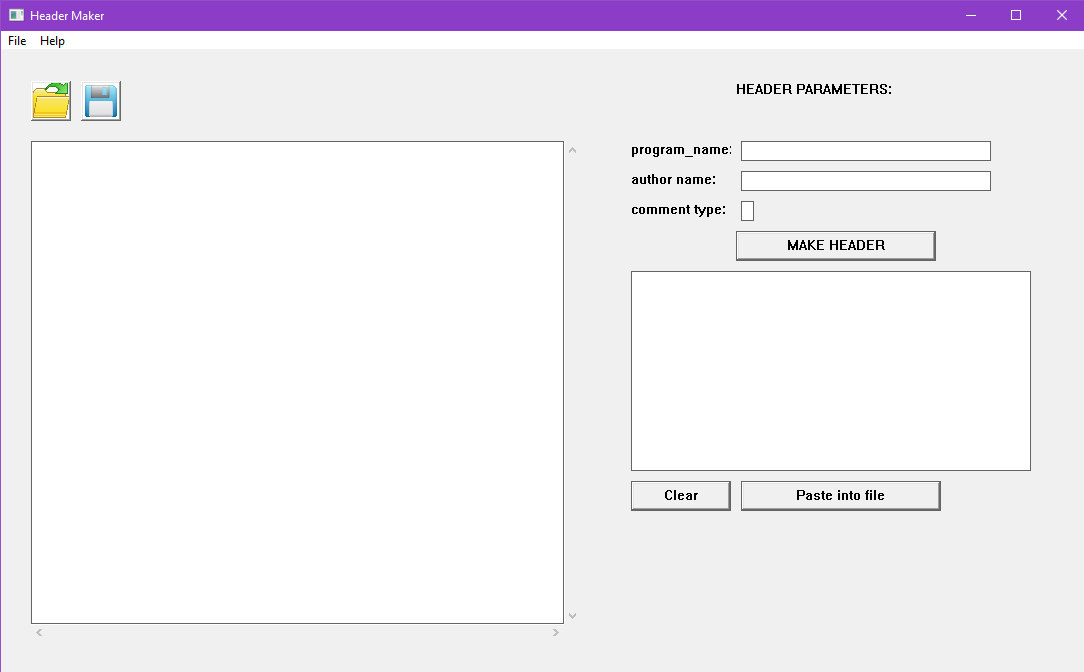
Розроблена програма на мові С++ має назву “Header Maker”. Вона створена за допомогою Windows API для генерування та вставки «шапок/хедерів» у текстові файли. Вигляд стартового вікна програми на рисунку 3.1.

Рисунок 3.1 – Стартове вікно програми

Стартове вікно містить в собі функціонал, що дозволяє створювати хедери та працювати з текстовими файлами: головне меню у лівому верхньому куті вікна, кнопки «Відкрити файл» та «Зберегти файл», поля вводу/виводу та кнопки роботи з ними.

При написанні програми були використані наступні функції для зв’язку аплікації з ОС та обробкою дій користувача:

* LoadCursor – завантажує заданий ресурс курсора з (.EXE) файлу виконуваної програми;
* CreateWindow – створює перекриваюче, спливаюче або дочірнє вікно. Вона визначає клас, заголовок, стиль вікна і (необов'язково) вихідне положення і розмір вікна. Функція також визначає вікно одного з батьків або власника, якщо такі взагалі є, і меню вікна;
* GetMessage – функція забирає повідомлення з черги;
* TranslateMessage – переводить повідомлення формату віртуальних клавіш в повідомлення символи;
* DispatchMessage – пересилає повідомлення віконній процедурі;
* CreateMenu – створює меню;
* AppendMenu – додає в кінець заданого рядка меню, "випадаючого" меню, підменю або контекстного меню новий пункт;
* SetMenu – пов’язує нове меню з заданим вікном;
* LoadImage – завантажує ресурс значка, курсору або точкового малюнка;
* SendMessage – відправляє заздалегідь підготовлене сигнальне повідомлення вікну або вікнам;
* SetWindowText – змінює текст рядка заголовка заданого вікна (якщо такий є). Якщо визначається вікно – елемент управління, то його текст змінюється;
* GetOpenFileName – створює стандартне діалогове вікно «Відкрити» (Open), яке дає можливість користувачеві визначити диск, каталог і ім'я файлу або ряд файлів, щоб відкрити;
* GetWindowTextLength – повертає довжину, в символах, тексту рядка заголовка визначеного вікна (якщо вікно має область заголовка);
* GetSaveFileName – створює стандартне діалогове вікно «Зберегти» (Save), яке дозволяє користувачеві визначити місце та ім’я файлу.

Команди, що були використані для функціонування процедур: WM\_COMMAND, WM\_CREATE, WM\_DESTROY.

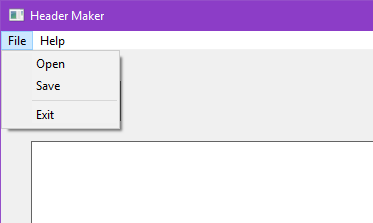
Головне меню, що складається з кнопок “File” i “Help”. При натисканні на “File” з’являється випадаюче меню (рис. 3.2), в якому є функції «відрикрити файл», «зберегти файл» та «вийти з програми».

Рисунок 3.2 – Випадаюче меню “File”

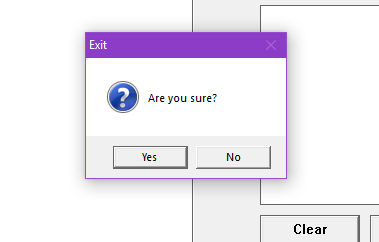
При натисканні на клавішу “Exit” для виходу з програми з’явиться спливаюче вікно, в якому користувача буде запитано чи впевнений він, що хоче закрити програму (рис. 3.3).

Рисунок 3.3 – Спливаюче вікно з запитом

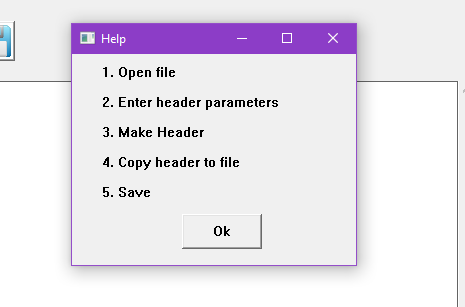
При натисканні на клавішу “Help” для отримання допомоги з роботою програми з’явиться нове вікно, в якому користувачеві буде запропонований перелік дій для зручного використання (рис. 3.4).

Рисунок 3.4 – Спливаюче вікно для допомоги з використанням програми

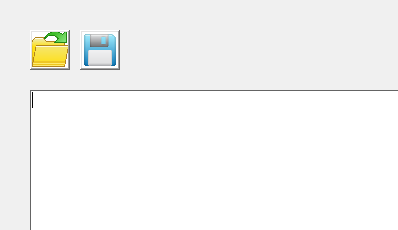
Кнопки швидкого доступу, що дозволяють виконати операції відкриття та збереження файлу (рис. 3.5).

Рисунок 3.5 – Кнопки швидкого доступу

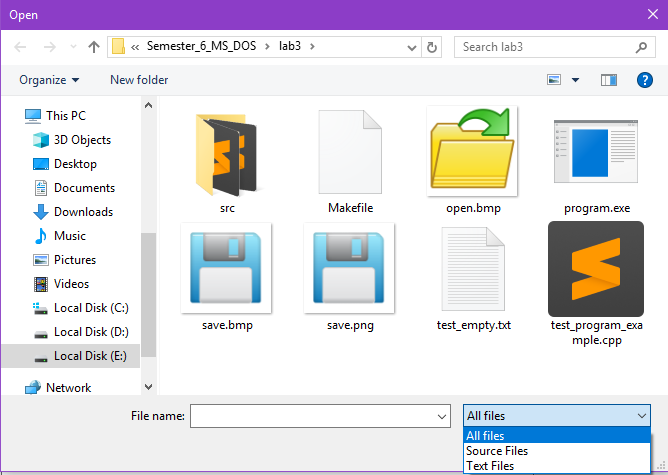
При натисканні на кнопку «відкрити файл» з’явиться відповідне меню (рис. 3.6), в якому можна встановити фільтр за розширенням та вибрати певний файл. Таке саме меню буде і при збереженні файлу. При натисканні кнопки «зберегти файл» користувачу буде запропоновано ввести ім’я нового файлу або існуючого файлу. У другому випадку попередня версія файлу буде змінена на ту, що знаходиться в полі вмісту файлу (біле поле під кнопками швидкого доступу). Якщо поле пусте, то файл після збереження стане пустим.

Рисунок 3.6 – Меню відкриття та збереження файлу

При відкритті файлу його зміст з’явиться у полі вмісту файлу. Оскільки кількість рядків та їх ширина може бути більше за параметри поля, то в такому випадку стануть активними кнопки прокрутки вверх-вниз та вправо-вліво (рис. 3.7). Кнопками прокрутки можна керувати за допомогою натискання миші на стрілочки або колеса миші.

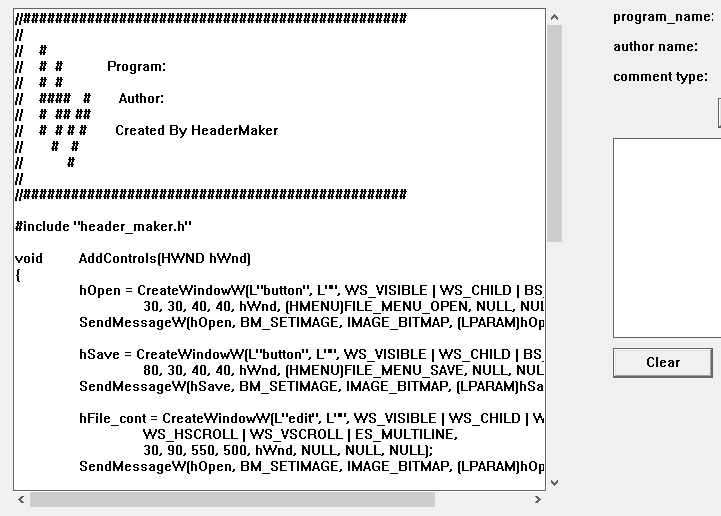
 В полі вмісту файлу можна редагувати файл, тому програма здатна виконувати базові функції тестового редактора.

Рисунок 3.7 – Робота з полем вмісту файлу

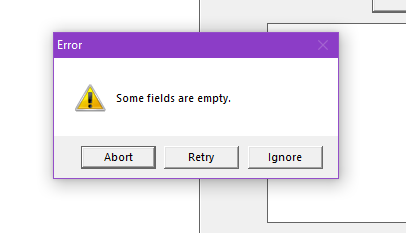
 Для того, щоб згенерувати хедер, необхідно ввести параметри: назва програми, автор програми, вид коментарю, що залежить від вибору мови програмування та натиснути кнопку “Make Header”. Якщо хоча б одне поле залишиться незаповненим, з’явиться спливаюче вікно, в якому буде повідомлення про помилку та запропоновані дії: закрити програму, спробувати ще раз та ігнорувати пусті поля (рис. 3.8).

Рисунок 3.8 – Повідомлення про помилку

 При генерації хедера його зображення з’явиться у відповідному полі, що знаходиться нижче кнопки “Make Header” (рис. 3.9). В залежності від вибраних символів зображення може бути трохи зміщене, але у текстових редакторах воно буде виглядати рівно. Це пов’язано з юнікодом та шириною кожного символу.

Рисунок 3.9 – Зображення згенерованого хедера

Нижче поля хедера знаходяться кнопки «Очистити» та «Вставити у файл». При натисканні на кнопку «Очистити» видаляється вміст поля хедера. При натисканні на кнопку «Вставити у файл» згенерований хедер з’являється у полі вмісту файлу на самому початку. Між хедером на початку змісту файлу вставляється пустий рядок для більш гарного представлення.

# 4 РЕЗУЛЬТАТИ ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

Для того, щоб продемонструвати повний функціонал програми необхідно:

1. відкрити файл;
2. заповнити параметри хедера;
3. згенерувати хедер;
4. вставити хедер у файл;
5. зберегти файл;
6. перевірити файл у вбудованому в ОС текстовому редакторі.

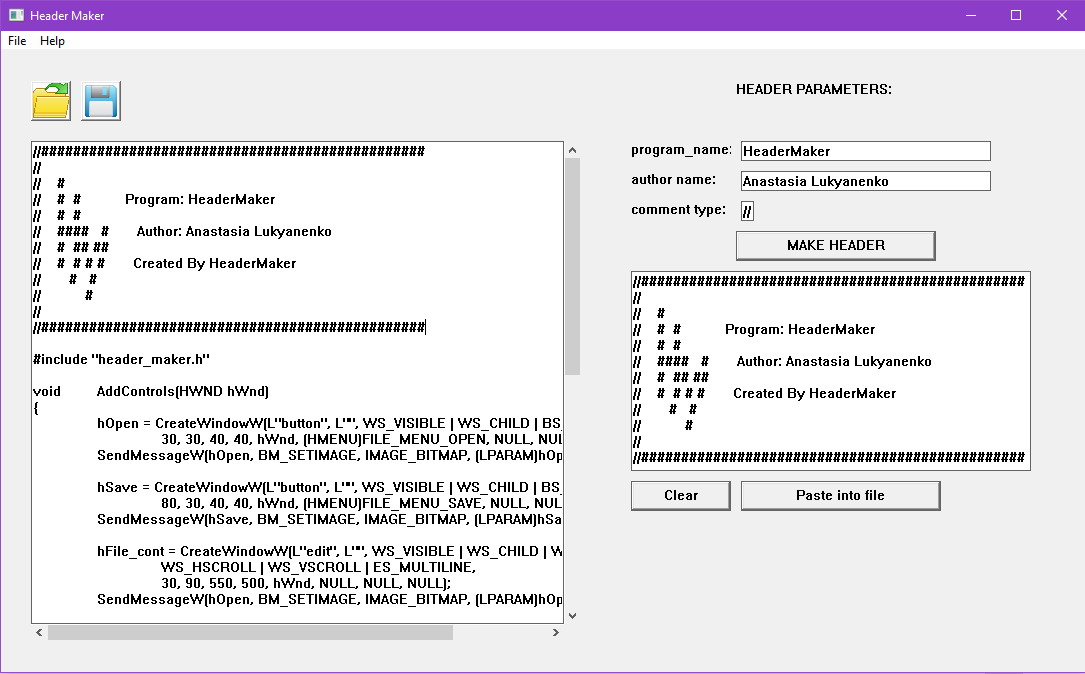
Окремі результати роботи програми наведені у третьому розділі. При вставці хедера у файл отримаємо поле, зображене на рисунку 4.1.

Рисунок 4.1 – Робота програми

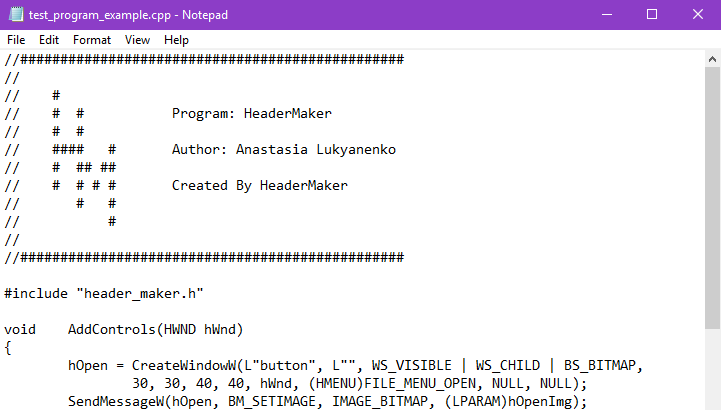
Після збереження результату необхідно перевірити файл у вбудованому в ОС текстовому редакторі, щоб переконатися, що файл відкривається, вміст файлу коректний та присутній хедер у правильному форматі. На рисунку 4.2 відредагований файл відкритий у текстовому редакторі Notepad.

Рисунок 4.2 – Перевірка файлу у текстовому редакторі

Таким чином розроблена програма правильно працює та виконує належні функції. Файл змінюється так, як того захоче користувач, помилок не виявлено.

# ВИСНОВКИ

У данiй лабораторнiй роботi була вивчена концепцiя повiдомлень в ОС Windows, отримані навички використання засобiв Windows API для роботи з повiдомленнями для поєднання рiзних компонентiв аплiкацiї або рiзних аплiкацiй мiж собою.

Була розроблена програма на мові С++ має назву “Header Maker”. Вона створена за допомогою Windows API для генерування та вставки «шапок/хедерів» у текстові файли.

Для розробки аплікації використані стандартнi повiдомлення Windows для уведення з клавiатури та за допомогою мишi, обмiну даними з елементами керування користувацького iнтерфейсу, реагування на команди меню, керування вiкнами, а також користувацькi повiдомлення для обмiну даними мiж компонентами аплiкацiї.

При розробці були використані тiльки засоби Windows API. Програма вiдлагоджена в ОС Windows 10 Pro.

# ПОСИЛАННЯ НА ЛIТЕРАТУРНI ДЖЕРЕЛА

1. Microsoft Library [Електронний ресурс]. — Режим доступу: <https://msdn.microsoft.com/en-us/library/>
2. Мешков А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. — [2-еизд.] — С.-Пб. : BHV, 2001. — 1040 с.
3. Petzold C. Programming Windows / C. Petzold. — [5th ed.] — Microsoft Press, 1998. — 1100 p.
4. Simon R. Windows NT Win32 API SuperBible / R. Simon. — Waite Group Press, 1997. — 1510 p.

# Додаток А Текст програми

Makefile

NAME = program

SRC = src/WindowProcedure.cpp \

src/open\_file.cpp \

src/save\_file.cpp \

src/header\_maker.cpp \

src/AddControls.cpp \

src/AddMenus.cpp \

src/dialog\_window.cpp

all: $(NAME)

$(NAME):

gcc $(SRC) -o $(NAME) -lcomdlg32 -lstdc++

Header

#ifndef HEADER\_MAKER\_H

#define HEADER\_MAKER\_H

# include <windows.h>

# include <stdlib.h>

# include <wchar.h>

# include <stdio.h>

# define FILE\_MENU\_OPEN 1

# define FILE\_MENU\_SAVE 2

# define FILE\_MENU\_EXIT 3

# define MAKE\_HEADER 4

# define CLEAR\_FIELD 5

# define FILE\_MENU\_HELP 6

# define PASTE\_INTO\_FILE 7

extern char \*prog\_name;

extern char \*author;

extern char \*comment\_type;

extern char \*header;

extern HWND hProg\_name;

extern HWND hAuthor;

extern HWND hComment\_type;

extern HWND hHeader;

extern HWND hFile\_cont;

extern HWND hSave;

extern HWND hOpen;

extern HMENU hMenu;

extern HBITMAP hSaveImg;

extern HBITMAP hOpenImg;

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp);

LRESULT CALLBACK DialogProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp);

void AddMenus(HWND);

void AddControls(HWND);

void loadImages();

void registerDialogClass(HINSTANCE hInst);

void displayDialog(HWND hWnd);

void open\_file(HWND hWnd);

void display\_file(char \*path);

void write\_file(char \*path);

void save\_file(HWND hWnd);

#endif

Main

#include "header\_maker.h"

char \*prog\_name;

char \*author;

char \*comment\_type;

char \*header;

HWND hProg\_name;

HWND hAuthor;

HWND hComment\_type;

HWND hHeader;

HWND hOpen;

HWND hSave;

HWND hFile\_cont;

HMENU hMenu;

HBITMAP hSaveImg;

HBITMAP hOpenImg;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow)

{

WNDCLASSW wc = {0};

wc.hbrBackground = (HBRUSH)COLOR\_WINDOW ;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hInstance = hInst;

wc.lpszClassName = L"myWindowClass";

wc.lpfnWndProc = WindowProcedure;

if (!(RegisterClassW(&wc)))

return -1;

registerDialogClass(hInst);

CreateWindowW(L"myWindowClass", L"Header Maker", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,

120, 35, 1100, 680, NULL, NULL, NULL, NULL);

MSG msg = {0};

while(GetMessage(&msg, NULL, NULL, NULL))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

Window Procedure

#include "header\_maker.h"

LRESULT CALLBACK DialogProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp)

{

switch(msg)

{

case WM\_COMMAND:

switch(wp)

{

case 1:

DestroyWindow(hWnd);

break;

}

break ;

case WM\_CLOSE:

DestroyWindow(hWnd);

break ;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp)

{

int i;

int j;

char \*str\_bord;

char \*str\_space;

int val;

switch (msg)

{

case WM\_COMMAND:

switch(wp)

{

case FILE\_MENU\_EXIT:

{

val = MessageBoxW(hWnd, L"Are you sure?", L"Exit",

MB\_YESNO | MB\_ICONQUESTION);

if (val == IDYES)

DestroyWindow(hWnd);

break;

}

case FILE\_MENU\_OPEN:

open\_file(hWnd);

break ;

case FILE\_MENU\_SAVE:

save\_file(hWnd);

break ;

case FILE\_MENU\_HELP:

displayDialog(hWnd);

break ;

case PASTE\_INTO\_FILE:

{

char \*file\_cont;

header = (char \*)malloc(sizeof(char) \* 999999);

file\_cont = (char \*)malloc(sizeof(char) \* 999999);

GetWindowText(hFile\_cont, file\_cont, 999999);

GetWindowText(hHeader, header, 999999);

strcat(header, "\r\n");

strcat(header, file\_cont);

SetWindowText(hFile\_cont, header);

break ;

}

case CLEAR\_FIELD:

{

SetWindowText(hHeader, "");

break ;

}

case MAKE\_HEADER:

i = 0;

prog\_name = (char \*)malloc(sizeof(char) \* 101);

author = (char \*)malloc(sizeof(char) \* 101);

comment\_type = (char \*)malloc(sizeof(char) \* 50);

header = (char \*)malloc(sizeof(char) \* 1001);

GetWindowText(hProg\_name, prog\_name, 100);

GetWindowText(hAuthor, author, 100);

GetWindowText(hComment\_type, comment\_type, 100);

if (strcmp(prog\_name, "") == 0 || strcmp(author, "") == 0

|| strcmp(comment\_type, "") == 0)

{

val = MessageBoxW(hWnd, L"Some fields are empty.", L"Error",

MB\_ABORTRETRYIGNORE | MB\_ICONEXCLAMATION);

switch(val)

{

case IDABORT:

DestroyWindow(hWnd);

break ;

case IDRETRY:

return 0;

case IDIGNORE:

break ;

}

}

strcpy(header, comment\_type);

str\_bord = (char \*)malloc(sizeof(char) \* 70);

str\_space = (char \*)malloc(sizeof(char) \* 70);

while (j < 48)

str\_bord[j++] = '#';

str\_bord[j] = '\0';

j = 0;

while (j < 69)

str\_space[j++] = ' ';

str\_space[j] = '\0';

while (i < 12)

{

if (i != 0)

strcat(header, comment\_type);

if (i == 0 || i == 11)

strcat(header, str\_bord);

if (i == 2)

strcat(header, " # ");

if (i == 3)

{

strcat(header, " # # ");

strcat(header, " Program: ");

strcat(header, prog\_name);

}

if (i == 4)

strcat(header, " # # ");

if (i == 5)

{

strcat(header, " #### # ");

strcat(header, " Author: ");

strcat(header, author);

}

if (i == 6)

strcat(header, " # ## ## ");

if (i == 7)

{

strcat(header, " # # # # ");

strcat(header, " Created By HeaderMaker ");

}

if (i == 8)

strcat(header, " # # ");

if (i == 9)

strcat(header, " # ");

strcat(header, "\r\n");

i++;

}

SetWindowText(hHeader, header);

free(str\_space);

free(str\_bord);

free(prog\_name);

free(author);

free(comment\_type);

free(header);

break ;

}

break ;

case WM\_CREATE:

loadImages();

AddMenus(hWnd);

AddControls(hWnd);

break ;

case WM\_DESTROY:

PostQuitMessage(0);

break ;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

}

Add Menu

#include "header\_maker.h"

void loadImages()

{

hOpenImg = (HBITMAP)LoadImageW(NULL, L"open.bmp", IMAGE\_BITMAP, 40, 40, LR\_LOADFROMFILE);

hSaveImg = (HBITMAP)LoadImageW(NULL, L"save.bmp", IMAGE\_BITMAP, 40, 40, LR\_LOADFROMFILE);

}

void AddMenus(HWND hWnd)

{

hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

AppendMenu(hFileMenu, MF\_STRING, FILE\_MENU\_OPEN, "Open");

AppendMenu(hFileMenu, MF\_STRING, FILE\_MENU\_SAVE, "Save");

AppendMenu(hFileMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(hFileMenu, MF\_STRING, FILE\_MENU\_EXIT, "Exit");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFileMenu, "File");

AppendMenu(hMenu, MF\_STRING, FILE\_MENU\_HELP, "Help");

SetMenu(hWnd, hMenu);

}