**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №3**

з дисципліни

«Системне програмування»

на тему

«Макровизначення і макроси в MASM32»

Виконав:

Перевірив:

студент групи ІП-93

Павлов Валерій Георгійович

Домінський Валентин Олексійович

номер залікової книжки: 9311

номер у списку: 9

Київ 2021

**Мета:**

Вивчити технології створення і вживання макросів. Дослідження результатів роботи макрогенератору MASM32

**Порядок виконання роботи:**

1. Вивчити правила використання макросів в програма
2. Переробити програму, підготовлену в лабораторній роботі 3 так, щоб виведення кожного виду інформації (дата народження, номер залікової книжки і т. п.) виконувалося у вигляді багатократного звернення до макросу (макрос 1), в якому у віконному інтерфейсі здійснювалося б виведення тексту, який передається в макрос як параметр

; Macros #1 for printing some text

PrintInformationInWindow macro heightPosition**,** infoToShow

; for example, this commentary is included into macroexpansion

;; but this - not

; just pass position of the text on vertical

; and text, that We want to show

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheText**,** offset infoToShow**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_TEXT **or** SS\_CENTER **or** BS\_VCENTER**,**

16**,** heightPosition**,** 170**,** 50**,**

hWnd**,** 7044**,** hInstance**,** NULL

endm

1. Оформити також у вигляді окремих макросів фрагменти програми, де здійснюється шифрування введеного рядка символів (макрос 2), а також порівняння її з хеш-кодом оригіналу пароля, що зберігається у програмі (макрос 3). Макровизначення для всіх макросів розмістити в тому ж файлі, що і програма. У всіх макросах обов'язково використовувати звичайні і приховані коментарі, а в останньому макросі також використовувати механізм оголошення локальних міток

; Macros #2 for decrypting string from user

DecryptStringFromUser macro StringFromUserInput

; for example, this commentary is included into macroexpansion

;; but this - not

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL LoopItself

; creating a loop, to check all letters

LoopItself**:**

;; incrementing counter edi

**inc** **edi**

; write one letter from input to ah register

**mov** **ah,** StringFromUser**[edi]**

;; decrypt one letter

**xor** **ah,** XORKey

; compare password length and counter

**cmp** **edi,** PasswordCount

;; if they are not equal, then continue the loop

**jne** LoopItself

endm

; Macros #3 for checking string from user

IsPasswordLegit macro StringFromUserInput

; for example, this commentary is included into macroexpansion

;; but this - not

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL WrongPassword

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL LoopItself

LoopItself**:**

; incrementing counter edi

**inc** **edi**

;; compare password length and register

**cmp** **ax,** PasswordCount

; if they are the same, then quit macros

**je** WrongPassword

;; write one letter from input to ah register

**mov** **ah,** StringFromUser**[edi]**

; check ah register and one letter from user's input

**cmp** **ah,** StringFromUserInput**[edi]**

;; if they are the same, then continue the loop

**je** LoopItself

WrongPassword**:**

; set some value, so our checks

;; will pass, IF password is legit

**mov** **ecx,** **-**10

endm

Увесь код:

; Processors

.386

.model **flat,** **stdcall**

option **CaseMap:None**

WinMainProto proto **:dword,:dword,:dword**

WinWarningProto proto **:dword,:dword,:dword**

WinFailureProto proto **:dword,:dword,:dword**

WinSuccessProto proto **:dword,:dword,:dword**

; Libraries And Macroses

include \masm32\include\windows.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

; Our Macroses

; We place them here, 'cause it won't degrade the readability of the code

; Macros #1 for printing some text

PrintInformationInWindow macro heightPosition**,** infoToShow

; for example, this commentary is included into macroexpansion

;; but this - not

; just pass position of the text on vertical

; and text, that We want to show

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheText**,** offset infoToShow**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_TEXT **or** SS\_CENTER **or** BS\_VCENTER**,**

16**,** heightPosition**,** 170**,** 50**,**

hWnd**,** 7044**,** hInstance**,** NULL

endm

; Macros #2 for decrypting string from user

DecryptStringFromUser macro StringFromUserInput

; for example, this commentary is included into macroexpansion

;; but this - not

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL LoopItself

; creating a loop, to check all letters

LoopItself**:**

;; incrementing counter edi

**inc** **edi**

; write one letter from input to ah register

**mov** **ah,** StringFromUser**[edi]**

;; decrypt one letter

**xor** **ah,** XORKey

; compare password length and counter

**cmp** **edi,** PasswordCount

;; if they are not equal, then continue the loop

**jne** LoopItself

endm

; Macros #3 for checking string from user

IsPasswordLegit macro StringFromUserInput

; for example, this commentary is included into macroexpansion

;; but this - not

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL WrongPassword

; create a local mark, to so that there won't be confusion,

;; when invoking macros more then 1 time

LOCAL LoopItself

LoopItself**:**

; incrementing counter edi

**inc** **edi**

;; compare password length and register

**cmp** **ax,** PasswordCount

; if they are the same, then quit macros

**je** WrongPassword

;; write one letter from input to ah register

**mov** **ah,** StringFromUser**[edi]**

; check ah register and one letter from user's input

**cmp** **ah,** StringFromUserInput**[edi]**

;; if they are the same, then continue the loop

**je** LoopItself

WrongPassword**:**

; set some value, so our checks

;; will pass, IF password is legit

**mov** **ecx,** **-**10

endm

.data?

hInstance HINSTANCE **?** ; Handle of our program

hWndOfMainWindow HWND **?** ; Handle of our main window

hWndOfWarnWindow HWND **?** ; Handle of our warn window

hWndOfSuccessWindow HWND **?** ; Handle of our success window

hWndOfFailureWindow HWND **?** ; Handle of our failure window

hWndOfEditbox HWND **?** ; Handle of our editbox

StringFromUser DB 128 dup**(?)**

; Data Segment

.data

StartingText DB "Введiть пароль у наступому вікні, щоб отримати дані"**,** 0

FailureText DB "Пароль невiрний. Спробуйте ще раз"**,** 0

; Name Of Message Box

MsgBoxName DB "4-9-IP93-Dominskyi"**,** 0

; We can write password in two ways:

Password DB "Mfd`gzbp`"

; And another one is:

; Password DB 31h 32h 33h

PasswordCount **=** **$-**Password

XORKey DB 9h

; Text To Show

InformationText DB "ПIБ = Домiнський Валентин Олексiйович"**,** 13**,**

"Дата Народження = 22.02.2002"**,** 13**,**

"Номер Залiковки = 9311"**,** 0

InformationTextSNP DB "ПIБ = Домiнський Валентин Олексiйович"**,** 0

InformationTextBirth DB "Дата Народження = 22.02.2002"**,** 0

InformationTextZalikova DB 13**,** "Номер Залiковки = 9311"**,** 0

NameOfTheStartingWindows DB "Window with starting text"**,** 0 ; the name of our window class

NameOfTheWarnWindows DB "Window with warn text"**,** 0 ; the name of our success window class

NameOfFailureWindows DB "Window with failure text"**,** 0 ; the name of our success window class

NameOfSuccessWindows DB "Window with some text"**,** 0 ; the name of our success window class

NameOfTheEditBox DB "Edit"**,** 0 ; the name of our editbox class

NameOfTheButton DB "Button"**,** 0 ; the name of our button class

NameOfTheText DB "Static"**,** 0 ; the name of our text class

TextForButton DB "Перевірити пароль"**,** 0

TextForOKButton DB "ОК"**,** 0

; Code Segment

.code

start**:** ; Generates program start-up code

invoke WinWarningProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

invoke GetModuleHandle**,** NULL

**mov** hInstance**,** **eax**

invoke WinMainProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

invoke ExitProcess**,** **eax** ; quit program. code returns in EAX register from Main Function.

; function declaration of WinMain

WinMainProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**2

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfTheStartingWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfTheStartingWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

470**,** 280**,** 300**,** 200**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfMainWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinMainProto endp

; function declaration of WinWarn

WinWarningProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndWarnProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**1

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfTheWarnWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfTheWarnWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

520**,** 310**,** 200**,** 150**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfWarnWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinWarningProto endp

; function declaration of WinSuccess

WinSuccessProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndSuccessProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**1

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfSuccessWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfSuccessWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

510**,** 280**,** 220**,** 200**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfSuccessWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinSuccessProto endp

; function declaration of WinSuccess

WinFailureProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndFailureProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**1

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfFailureWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfFailureWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

510**,** 280**,** 220**,** 150**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfFailureWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinFailureProto endp

WndSuccessProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

; invoke macros #1 three times to create text

PrintInformationInWindow 10**,** offset InformationTextSNP

PrintInformationInWindow 40**,** offset InformationTextBirth

PrintInformationInWindow 70**,** offset InformationTextZalikova

; create button

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForOKButton**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

65**,** 125**,** 70**,** 30**,**

hWnd**,** 7033**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,eax**

**ret**

WndSuccessProc endp

WndFailureProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

; invoke macros #1 one time to create text

PrintInformationInWindow 10**,** offset FailureText

; create button

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForOKButton**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

65**,** 65**,** 70**,** 30**,**

hWnd**,** 7033**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,** **eax**

**ret**

WndFailureProc endp

WndProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

; create editbox

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheEditBox**,** NULL**,**

WS\_CHILD **or** WS\_VISIBLE **or** ES\_LEFT **or** ES\_AUTOHSCROLL **or** ES\_AUTOVSCROLL **,**

65**,**20**,**150**,** 30**,**

hWnd**,** 7000**,** hInstance**,** NULL

**mov** hWndOfEditbox**,** **eax**

; create button

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForButton**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

60**,** 90**,** 170**,** 30**,**

hWnd**,** 7001**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

**mov** **bx,** 03h ; counter for tries

**cmp** wParam**,** 7001

**jne** ExitCode

; get text from editbox

invoke SendMessage**,** hWndOfEditbox**,** WM\_GETTEXT**,** PasswordCount**+**2**,** offset StringFromUser

; check password's length

**mov** **edi,** 0

; compare and if password's length is not the same, as origin...

**cmp** **ax,** PasswordCount

; ... jump to bad end

**jne** WrongPasswordByUser

; if length is equal to original password, then start checks

; there We have macros #2, which uses XOR to decrypt Our password

DecryptStringFromUser StringFromUser

; macros #3, where We check Our password

IsPasswordLegit StringFromUser

; if it's not equal, then password is legit

**cmp** **ecx,** 10

**jne** LegitPasswordByUser

; Unconditional jump to end

**jmp** WrongPasswordByUser

WrongPasswordByUser**:**

; counting tries

**add** **bx,** **-**01h ; decrementing

**cmp** **bx,** **-**01h ; negative possible tries

**je** TotalExitCode

invoke WinFailureProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

**jmp** ExitCode

LegitPasswordByUser**:**

invoke WinSuccessProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

**jmp** ExitCode

TotalExitCode**:**

invoke DestroyWindow**,**hWndOfMainWindow

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,eax**

**ret**

WndProc endp

WndWarnProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForOKButton**,**

WS\_CHILD **or** WS\_VISIBLE **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

55**,** 65**,** 70**,** 30**,**

hWnd**,** 7003**,** hInstance**,** NULL

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheText**,** offset StartingText**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_TEXT **or** SS\_CENTER **or** BS\_VCENTER**,**

16**,** 10**,** 150**,** 50**,**

hWnd**,** 7004**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,eax**

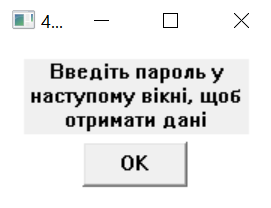
**ret**

WndWarnProc endp

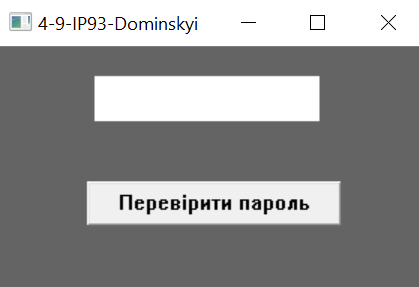
end start

1. Виконати компіляцію і компоновку файлу програми
2. Перевірити роботу програми шляхом введення як правильного, так і невірного паролів

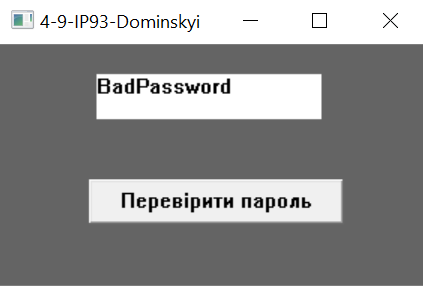
Початкове вікно:



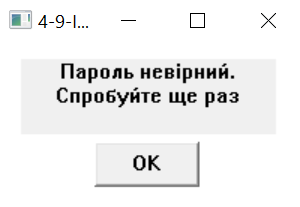
Вікно введення паролю:



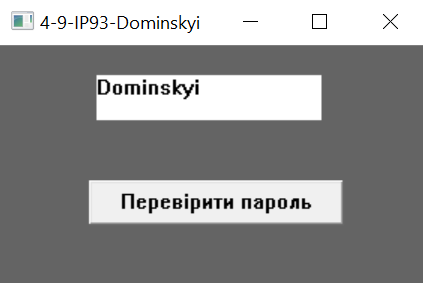
Введення неправильного паролю:



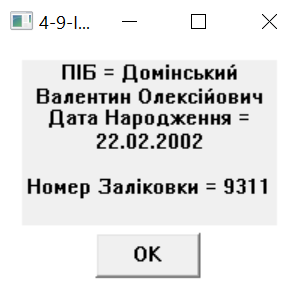
Вивід при такому паролі:



Введення правильно паролю:

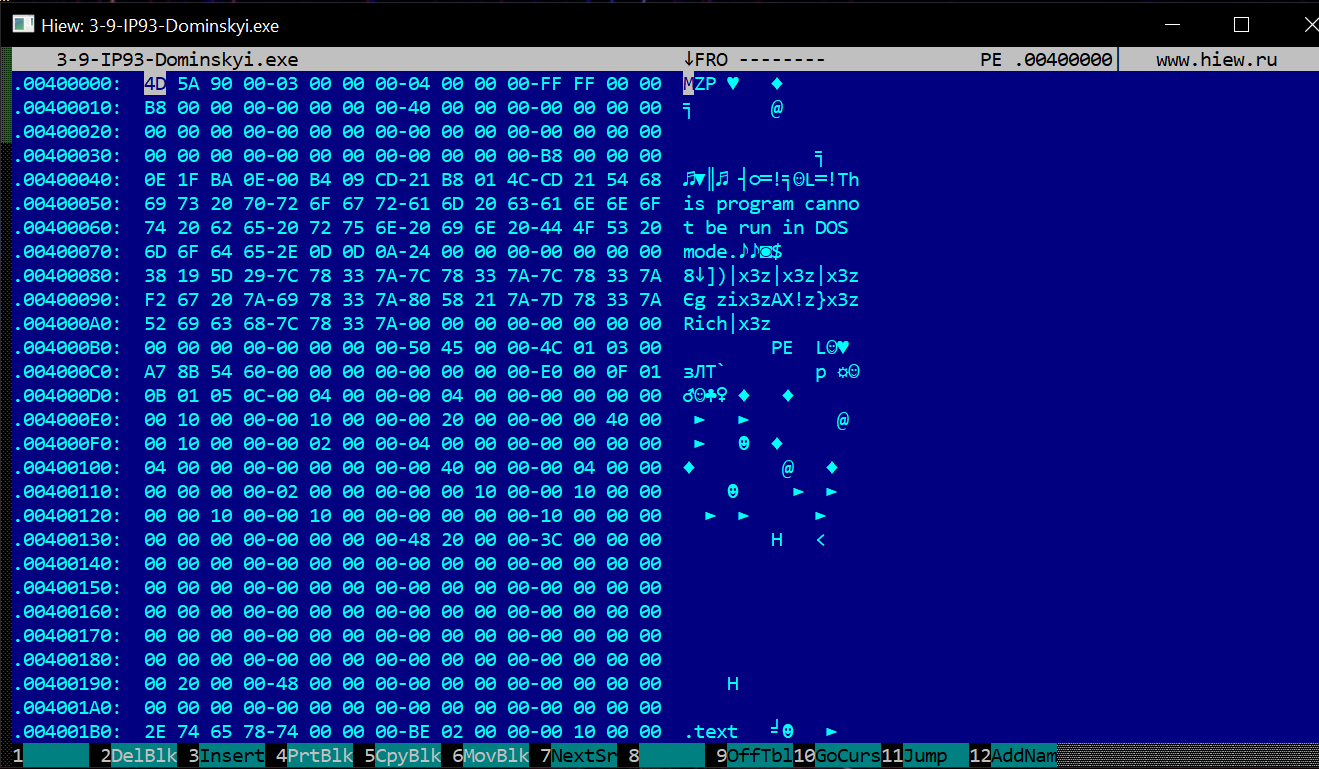


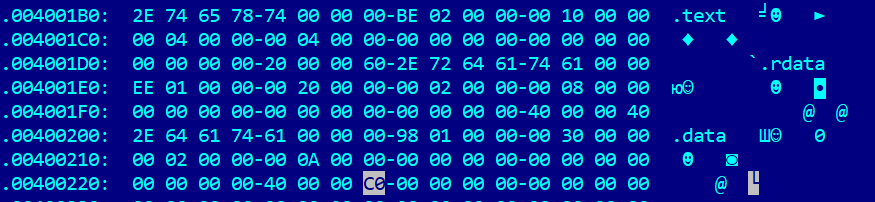
Вивід:



1. Отримати розширений лістинг програми за допомогою опції /Fl компілятора ML
2. Провести дослідження отриманого лістингу: визначити, яким чином компілятор виконав трансляцію кожній з команд макросів і помітити їх в тексті розширеного лістингу кольоровими олівцями або кольоровими фломастерами:
3. На скриншоті перших 25 рядків вмісту файлу обвести кольоровим олівцем або фломастером області MS-DOS заголовка (DOS\_HEADER), PE заголовка (PE\_HEADER) і таблиці секцій (SECTION\_HEADERS). Скріншот привести в звіті по лабораторній роботі, де:

* Червоний – MS-DOS заголовок
* Синій - Machine Type
* Жовтий – PE заголовок
* Білий - Optional Header Standard Fields
* Зелений - таблиці секцій





1. Відповідно до опису секцій скласти таблицю, в яку занести параметри свого файлу, вказані в розділах 3.3.1, 3.4.1 і 4
2. У останньому стовпчику таблиці розшифрувати виписані значення полів заголовка файлу. Таблицю привести в звіті по лабораторній роботі

|  |  |  |
| --- | --- | --- |
| Поле | Значення | Опис |
| **Machine Types** | | |
| IMAGE\_FILE\_MACHINE\_I386 | 0x14c  01 4C | Intel 386 or later, and compatible processors. |
| **Optional Header Standard Fields** | | |
| Magic | 01 0B | Unsigned integer identifying the state of the  image file. The most common number is  0413 octal (0x10B), identifying it as a  normal executable file. 0407 (0x107)  identifies a ROM image |
| MajorLinkerVersion | 05 | Linker major version number |
| MinorLinkerVersion | 0C | Linker minor version number |
| SizeOfCode | 00 04 00 00 | Size of the code (text) section, or the sum of  all code sections if there are multiple  sections. |
| SizeOfInitializedData | 00 04 00 00 | Size of the initialized data section, or the  sum of all such sections if there are multiple  data sections. |
| SizeOfUninitializedData | 00 00 00 00 | Size of the uninitialized data section (BSS),  or the sum of all such sections if there are  multiple BSS sections. |
| AddressOfEntryPoint | 00 01 00 00 | Address of entry point, relative to image  base, when executable file is loaded into  memory. For program images, this is the  starting address. For device drivers, this is  the address of the initialization function. An  entry point is optional for DLLs. When none  is present this field should be 0 |
| BaseOfCode | 00 10 00 00 | Address, relative to image base, of  beginning of code section, when loaded into  memory |
| BaseOfData | 00 20 00 00 | Address, relative to image base, of  beginning of data section, when loaded into  memory |
| **Section Table (Section Headers)** | | |
| Name | 00 00 00 74  78 65 74 2E | An 8-byte, null-padded ASCII string. There is no  terminating null if the string is exactly eight  characters long. For longer names, this field  contains a slash (/) followed by ASCII representation  of a decimal number: this number is an offset into  the string table. Executable images do not use a  string table and do not support section names longer  than eight characters. Long names in object files will  be truncated if emitted to an executable file. |
| VirtualSize | 00 00 02 BE | Total size of the section when loaded into memory.  If this value is greater than Size of Raw Data, the  section is zero-padded. This field is valid only for  executable images and should be set to 0 for object  files |
| VirtualAddress | 00 00 10 00 | For executable images this is the address of the first  byte of the section, when loaded into memory,  relative to the image base. For object files, this field  is the address of the first byte before relocation is  applied; for simplicity, compilers should set this to  zero. Otherwise, it is an arbitrary value that is  subtracted from offsets during relocation. |
| SizeOfRawData | 00 00 04 00 | Size of the section (object file) or size of the  initialized data on disk (image files). For executable  image, this must be a multiple of FileAlignment from  the optional header. If this is less than VirtualSize  the remainder of the section is zero filled. Because  this field is rounded while the VirtualSize field is not  it is possible for this to be greater than VirtualSize as  well. When a section contains only uninitialized data,  this field should be 0 |
| PointerToRawData | 00 00 04 00 | File pointer to section’s first page within the COFF  file. For executable images, this must be a multiple  of FileAlignment from the optional header. For  object files, the value should be aligned on a fourbyte boundary for best performance. When a section  contains only uninitialized data, this field should be  0 |
| PointerToRelocations | 00 00 00 00 | File pointer to beginning of relocation entries for the  section. Set to 0 for executable images or if there  are no relocations |
| PointerToLinenumbers | 00 00 00 00 | File pointer to beginning of line-number entries for  the section. Set to 0 if there are no COFF line  numbers |
| NumberOfRelocations | 00 00 | Number of relocation entries for the section. Set to 0  for executable images |
| NumberOfLinenumbers | 00 00 | Number of line-number entries for the section |
| Characteristics | 60 00 00 20 | Flags describing section’s characteristics |

1. Провести дослідження того ж файлу за допомогою меню "PE Editor" безкоштовної програми PE Tools. Всі скріншоти вікон програми з даними, відповідними раніше побудованій таблиці, привести в звіті по лабораторній роботі

Головна сторінка:

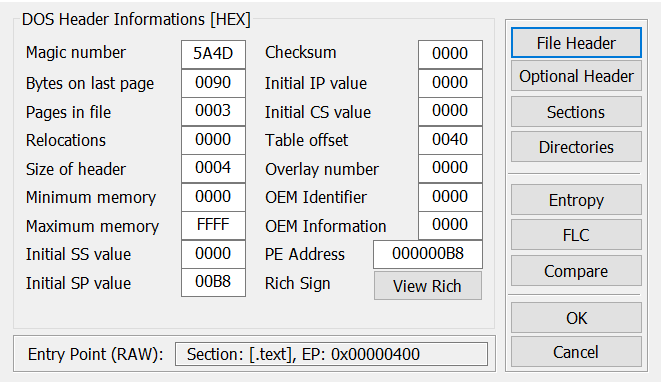


Image File Header Editor:

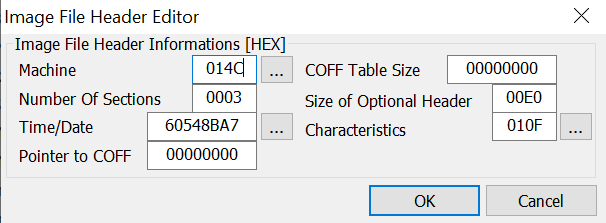
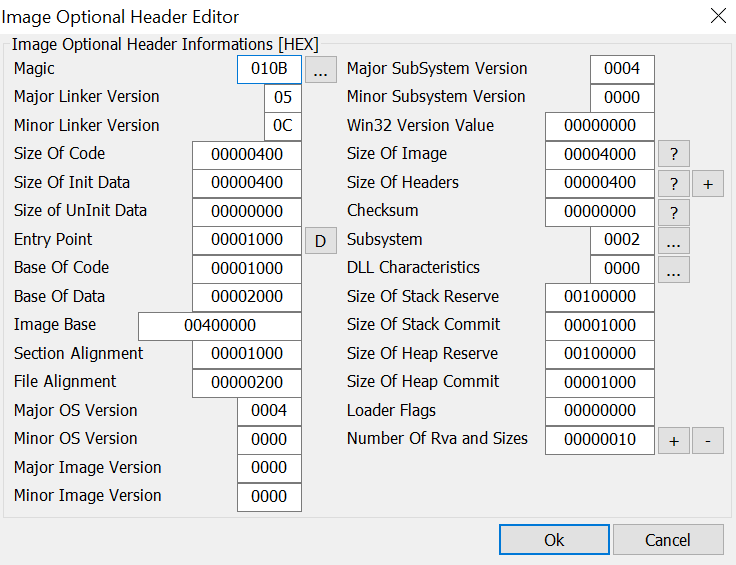
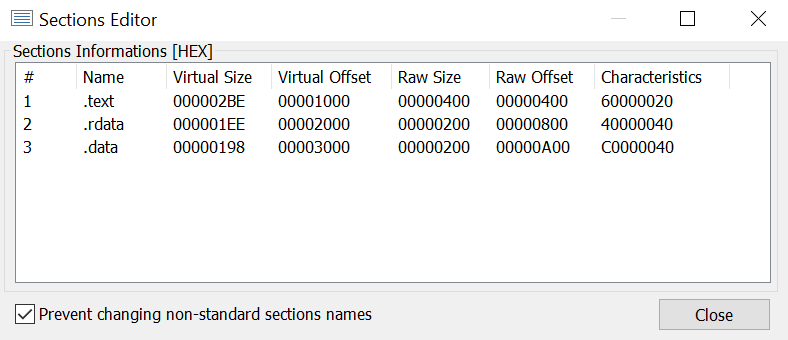


Image Optional Header Editor:

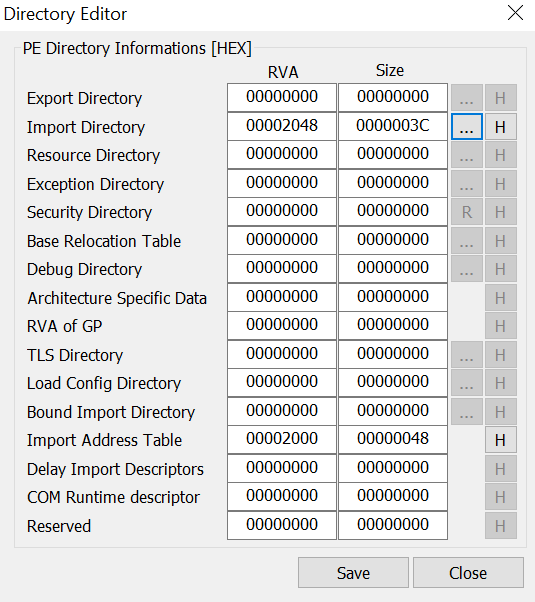


Sections Editor:

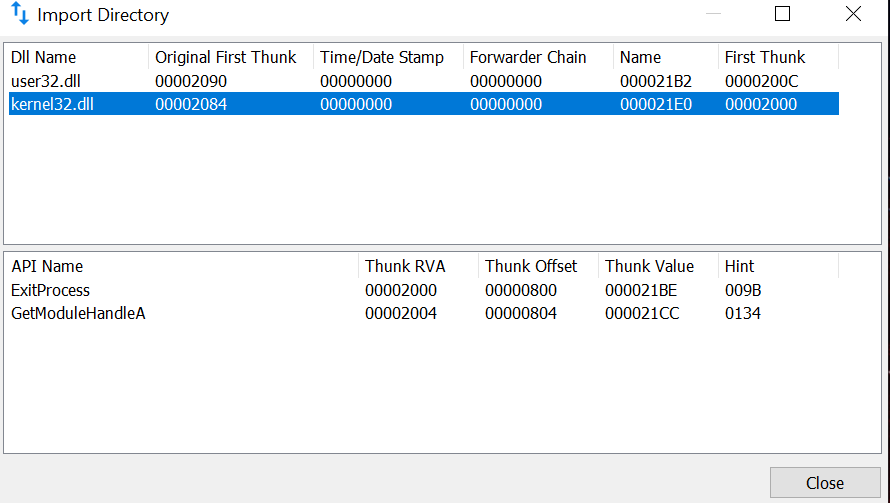
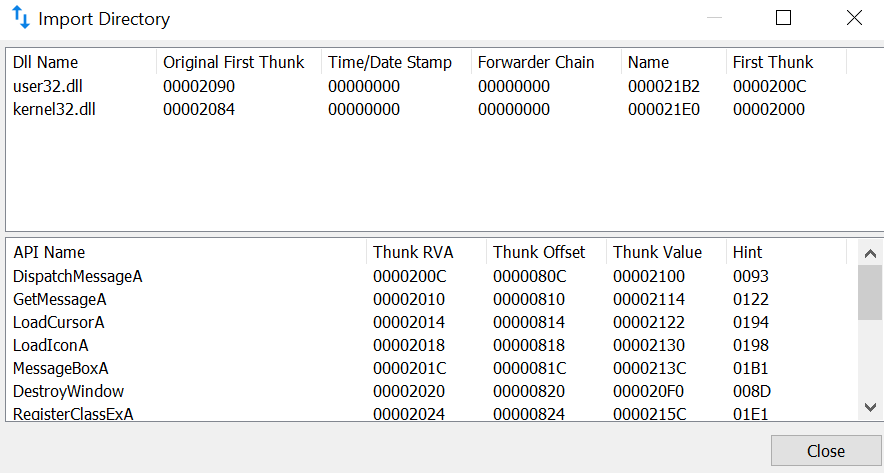


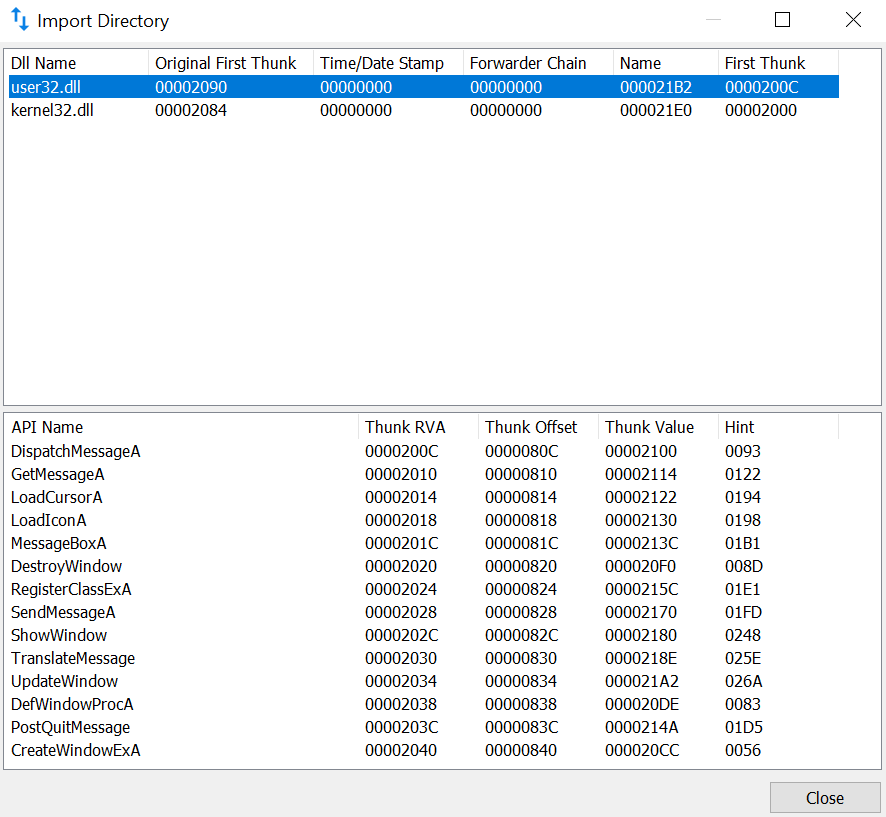
1. Дослідити таблицю імпорту (Import Directory) даного файлу і визначити, які саме функції використовуються з бібліотек, що підключаються. Скріншоти вікон Import Directory з функціями, що імпортуються, з кожного бібліотечного файлу привести в звіті по лабораторній роботі

Directory Editor:



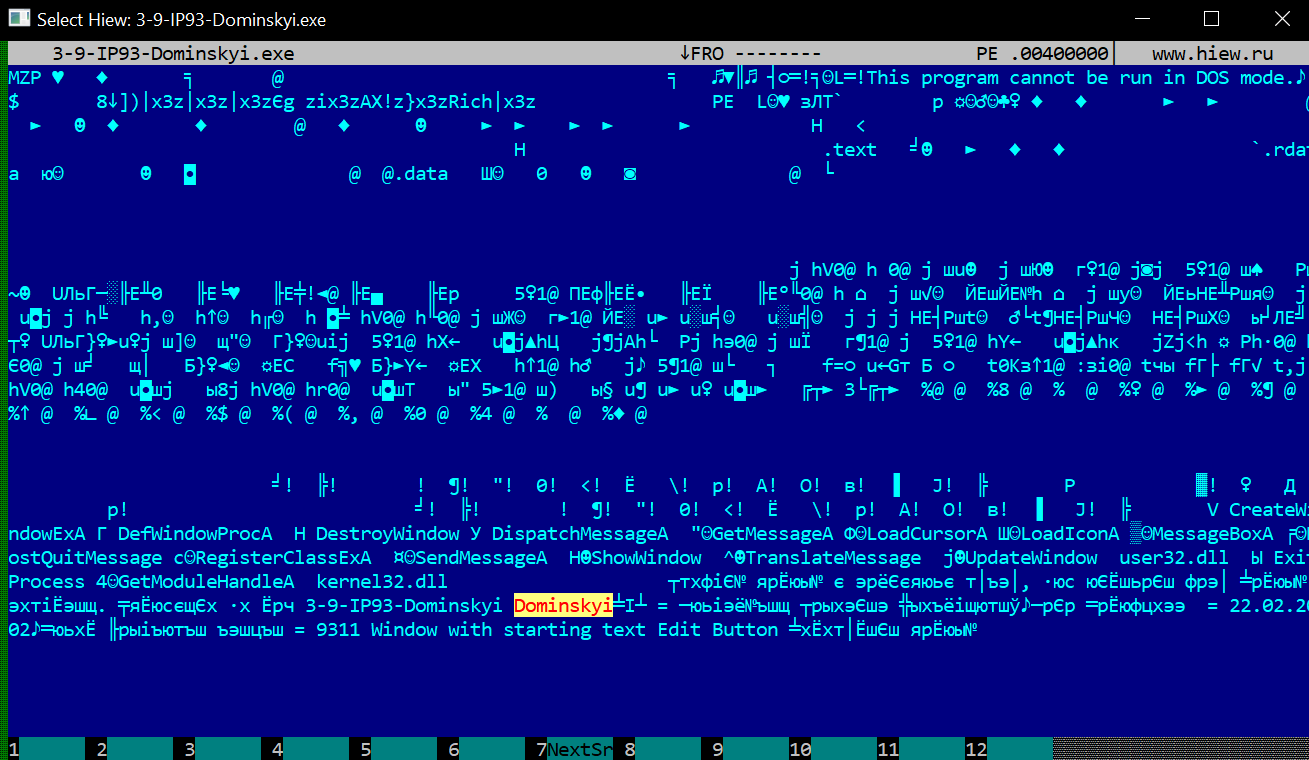
Бібліотеки:



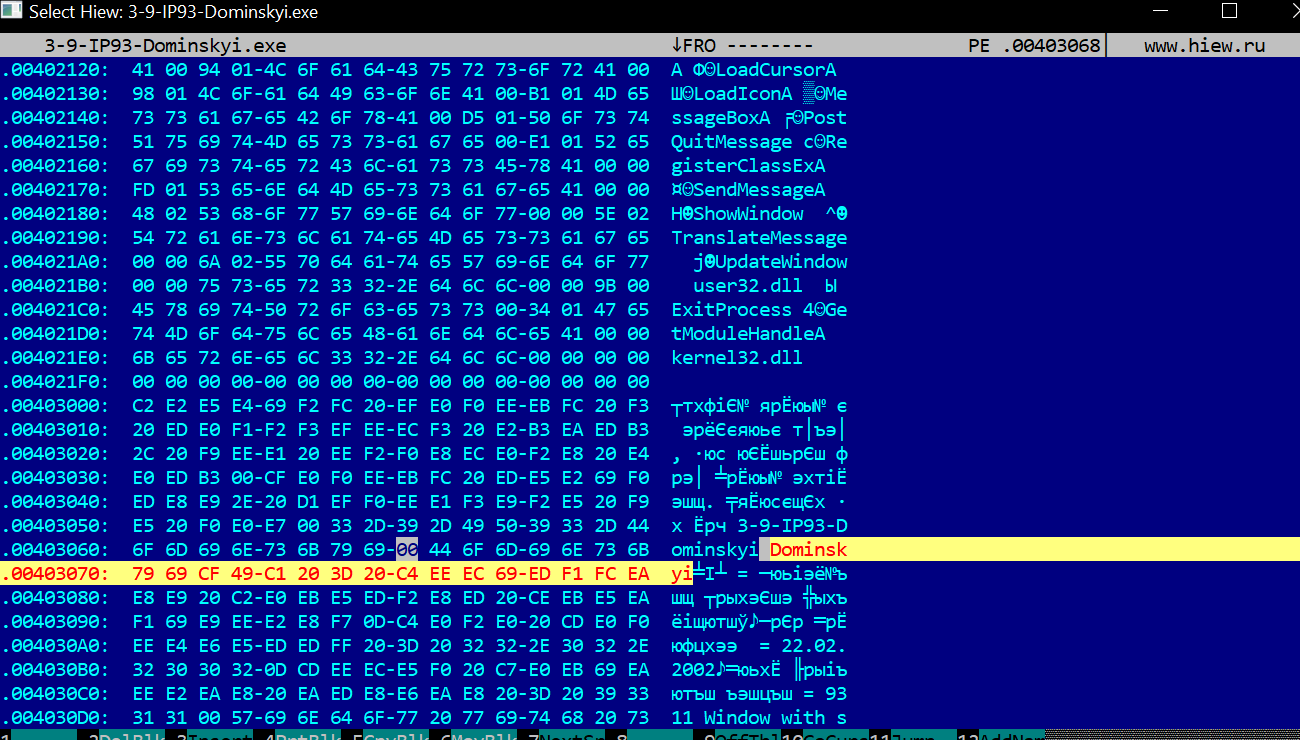


1. Знайти в тексті файлу по зсуву, узятому з побудованої таблиці, секцію з даними і переконатися, що текст оригінала пароля, що міститься в тексті програми, може бути легко виявлений за допомогою HEX-редактора. Привести скриншот цього фрагмента програми у вигляді HEX - коду в звіті по лабораторній роботі:

Text:



HEX:



1. Виконати шифрування пароля за допомогою функції XOR, знову скомпілювати EXE - файл і переконатися, що тепер вони не виявляються явним чином в тексті виконуваного EXE - файлу. Привести скріншоти цієї програми в режимах «Hex» і «Text» в звіті по лабораторній роботі:

; Processors

.386

.model **flat,** **stdcall**

option **CaseMap:None**

WinMainProto proto **:dword,:dword,:dword**

; Libraries And Macroses

include \masm32\include\windows.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

.data?

hInstance HINSTANCE **?** ; Handle of our program

hWndOfMainWindow HWND **?** ; Handle of our editbox

hWndOfEditbox HWND **?** ; Handle of our editbox

StringFromUser DB 128 dup**(?)**

; Data Segment

.data

StartingText DB "Введiть пароль у наступому вікні, щоб отримати дані"**,** 0

FailureText DB "Пароль невiрний. Спробуйте ще раз"**,** 0

; Name Of Message Box

MsgBoxName DB "3-9-IP93-Dominskyi"**,** 0

; We can write password in two ways:

Password DB "Mfd`gzbp`"

; And another one is:

; Password DB 31h 32h 33h

PasswordCount **=** **$-**Password

XORKey DB 9h

; Text To Show

InformationText DB "ПIБ = Домiнський Валентин Олексiйович"**,** 13**,**

"Дата Народження = 22.02.2002"**,** 13**,**

"Номер Залiковки книжки = 9311"**,** 0

NameOfTheStartingWindows DB "Window with starting text"**,** 0 ; the name of our window class

NameOfTheEditBox DB "Edit"**,** 0 ; the name of our editbox class

NameOfTheButton DB "Button"**,** 0 ; the name of our button class

TextForButton DB "Перевірити пароль"**,** 0

; Code Segment

.code

start**:** ; Generates program start-up code

invoke MessageBox**,** 0**,** offset StartingText**,** offset MsgBoxName**,** MB\_OK

invoke GetModuleHandle**,** NULL

**mov** hInstance**,** **eax**

invoke WinMainProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

invoke ExitProcess**,** **eax** ; quit program. code returns in EAX register from Main Function.

; function declaration of WinMain

WinMainProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need local variables

local wc**:**WNDCLASSEX

local msg**:**MSG

local hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**2

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfTheStartingWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfTheStartingWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

470**,** 280**,** 300**,** 200**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfMainWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE if WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .if **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinMainProto endp

WndProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.if ourMSG**==**WM\_CLOSE

; exit program

invoke PostQuitMessage**,**NULL

.elseif ourMSG**==**WM\_CREATE

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheEditBox**,** NULL**,**

WS\_VISIBLE **or** WS\_CHILD **or** ES\_LEFT **or** ES\_AUTOHSCROLL **or** ES\_AUTOVSCROLL **,**

65**,**20**,**150**,** 30**,**

hWnd**,** 7000**,** hInstance**,** NULL

**mov** hWndOfEditbox**,** **eax**

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForButton**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

60**,** 90**,** 170**,** 30**,**

hWnd**,** 7001**,** hInstance**,** NULL

.elseif ourMSG**==**WM\_COMMAND

**mov** **bx,** 03h ; counter for tries

**cmp** wParam**,** 7001

**jne** ExitCode

invoke SendMessage**,** hWndOfEditbox**,** WM\_GETTEXT**,** PasswordCount**+**2**,** offset StringFromUser

**mov** **edi,** 0

**cmp** **ax,** PasswordCount

**jne** WrongPasswordByUser

LoopItself**:**

**inc** **edi** ; incrementing

**loop** IsPasswordCorrect

IsPasswordCorrect**:**

**cmp** **edi,** PasswordCount

**je** CorrectPasswordByUser

**mov** **ah,** StringFromUser**[edi]**

**xor** **ah,** XORKey

**cmp** **ah,** Password**[edi]** ; Compare

**je** LoopItself ; Jump Equal

**jmp** WrongPasswordByUser ; Unconditional jump

WrongPasswordByUser**:**

; counting tries

**add** **bx,** **-**01h ; decrementing

**cmp** **bx,** **-**01h ; negative possible tries

**je** TotalExitCode

invoke MessageBox**,** hWnd**,** offset FailureText**,** offset MsgBoxName**,** MB\_OK

**jmp** ExitCode

CorrectPasswordByUser**:**

invoke MessageBox**,** hWnd**,** offset InformationText**,** offset MsgBoxName**,** MB\_OK

**jmp** ExitCode

TotalExitCode**:**

invoke DestroyWindow**,**hWndOfMainWindow

.else

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,eax**

**ret**

WndProc endp

end start

Шифрування паролю:

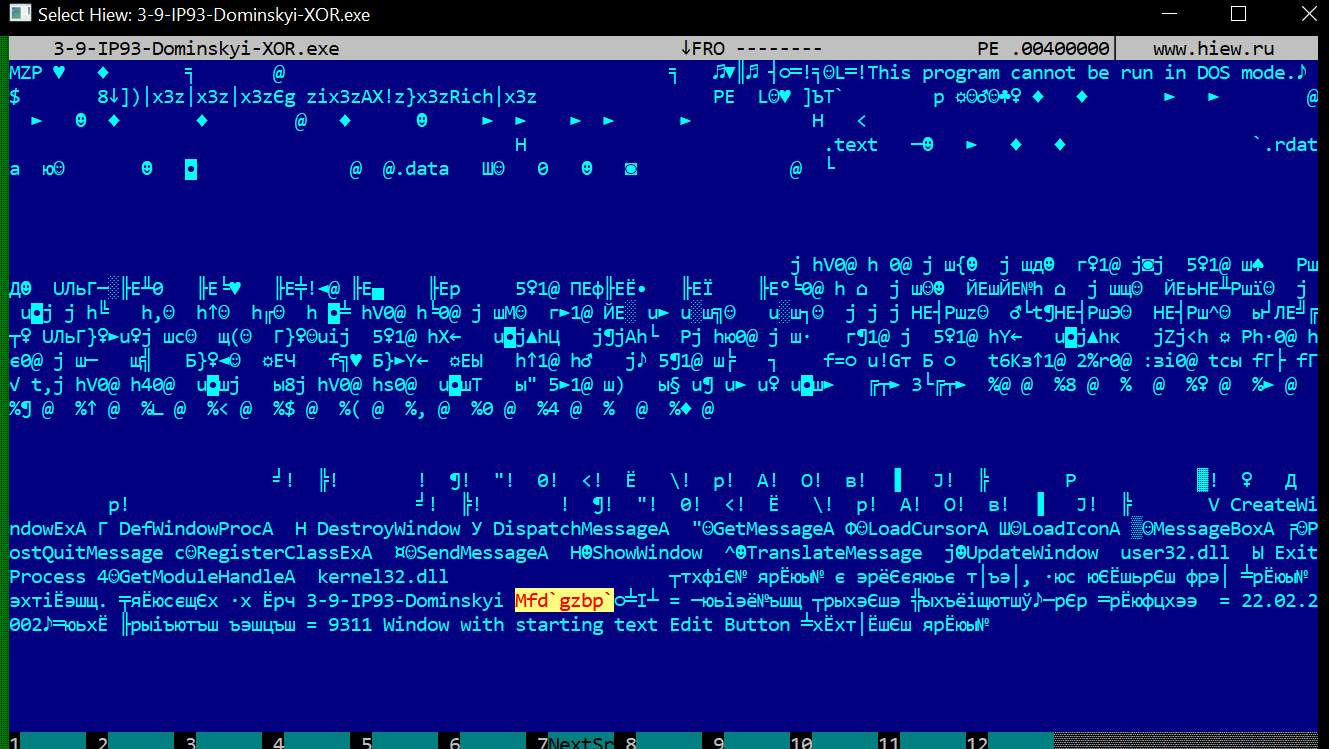
Ключ – 9 (номер у списку)

Пароль до шифрування – Dominskyi

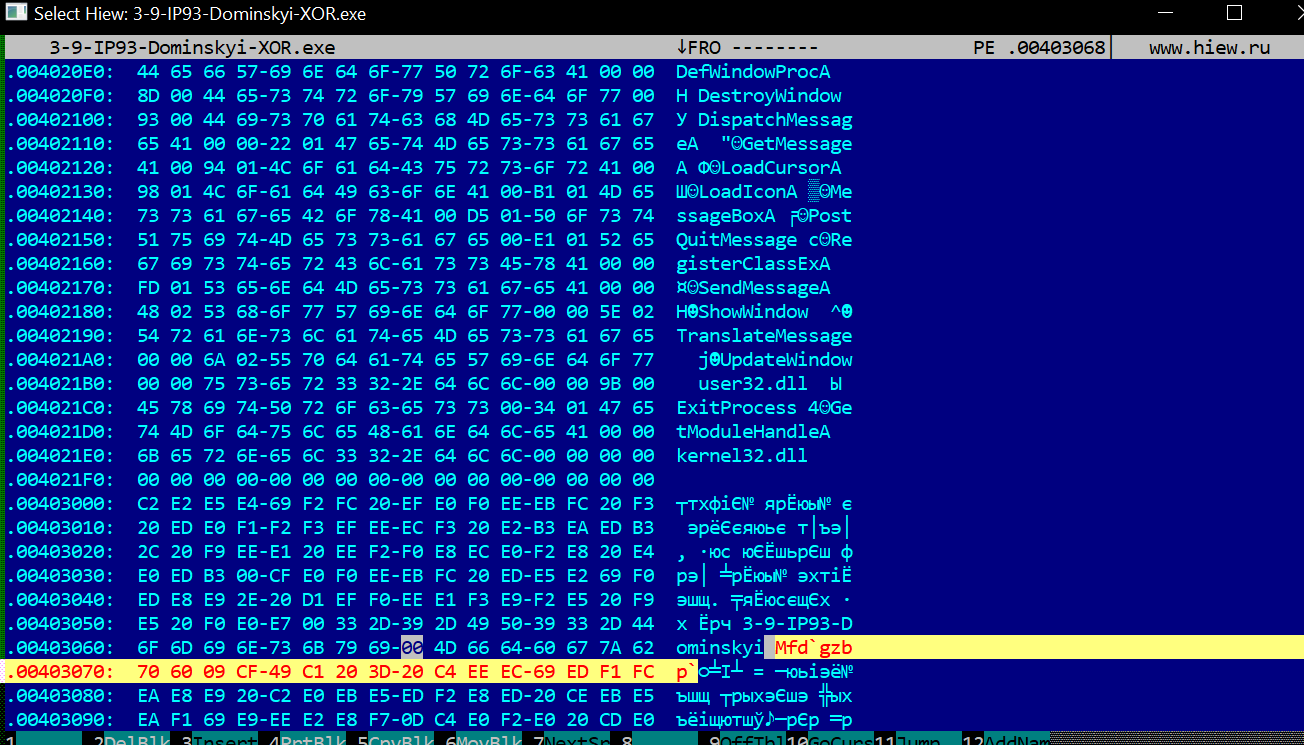
Пароль після шифрування - Mfd`gzbp`

* D -> 0100 0100 -> 0100 1101 -> M
* o -> 0110 1111 -> 0110 0110 -> f
* m -> 0110 1101 -> 0110 0100 -> d
* i -> 0110 1001 -> 0110 0000 -> `
* n -> 0110 1110 -> 0110 0111 -> g
* s -> 0111 0011 -> 0111 1010 -> z
* k -> 0110 1011 -> 0110 0010 -> b
* y -> 0111 1001 -> 0111 0000 -> p
* i -> 0110 1001 -> 0110 0000 -> `

Text:



HEX:



1. Зробити висновки по лабораторній роботі:

**Висновок:**

Я навчився працювати з різними віконними інтерфейсами та форматом EXE в середовищі Masm32. Закріпив на практиці шифрування та розшифрування паролі. Використав програму HIEW для читання лістингу та провів дослідження файлу за допомогою PE-Editor