**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 №4**

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**

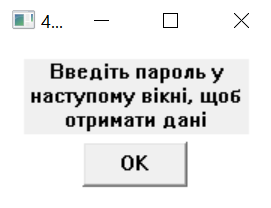
**ret**

WndWarnProc endp

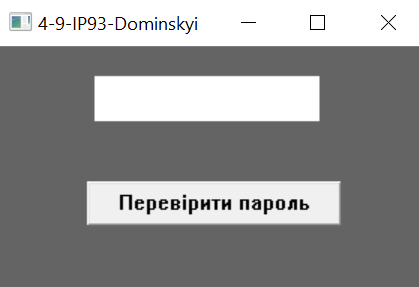
end start

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

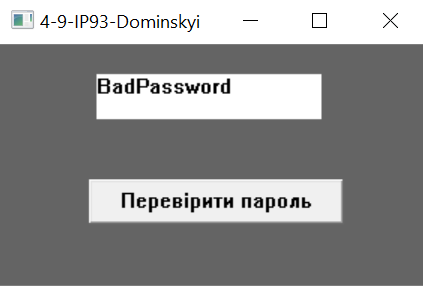
Початкове вікно:



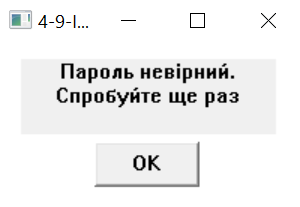
Вікно введення паролю:



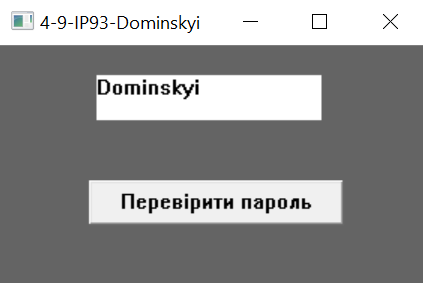
Введення неправильного паролю:



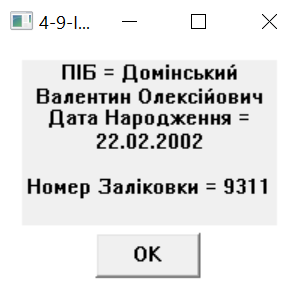
Вивід при такому паролі:



Введення правильного 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:

Сірий – перший макрос з виведенням тексту

Бірюзовий – другий макрос з розшифруванням паролю

Зелений – третій макрос з перевіркою паролю

**;** 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

00000000 **.data?**

00000000 00000000 hInstance HINSTANCE **?** **;** Handle of our **program**

00000004 00000000 hWndOfMainWindow HWND **?** **;** Handle of our main window

00000008 00000000 hWndOfWarnWindow HWND **?** **;** Handle of our warn window

0000000C 00000000 hWndOfSuccessWindow HWND **?** **;** Handle of our success window

00000010 00000000 hWndOfFailureWindow HWND **?** **;** Handle of our failure window

00000014 00000000 hWndOfEditbox HWND **?** **;** Handle of our editbox

00000018 00000080 **[** StringFromUser DB 128 dup**(?)**

00

**]**

**;** **Data** Segment

00000000 **.data**

00000000 C2 E2 E5 E4 69 StartingText DB "┬тхфiЄ№ ярЁюы№ є эрёЄєяюьє т│ъэ│, ∙юс юЄЁшьрЄш фрэ│"**,** 0

F2 FC 20 EF

E0 F0 EE EB

FC 20 F3 20

ED E0 F1 F2

F3 EF EE EC

F3 20 E2 B3

EA ED B3 2C

20 F9 EE E1

20 EE F2 F0

E8 EC E0 F2

E8 20 E4 E0

ED B3 00

00000034 CF E0 F0 EE EB FailureText DB "╧рЁюы№ эхтiЁэшщ. ╤яЁюсєщЄх ∙х Ёрч"**,** 0

FC 20 ED E5

E2 69 F0 ED

E8 E9 2E 20

D1 EF F0 EE

E1 F3 E9 F2

E5 20 F9 E5

20 F0 E0 E7

00

**;** Name Of Message Box

00000056 34 2D 39 2D 49 MsgBoxName DB "4-9-IP93-Dominskyi"**,** 0

50 39 33 2D

44 6F 6D 69

6E 73 6B 79

69 00

**;** We can **write** password in two ways**:**

00000069 4D 66 64 60 67 Password DB "Mfd`gzbp`"

7A 62 70 60

00000072

**;** And another one is**:**

**;** Password DB 31h 32h 33h

**=** 00000009 PasswordCount **=** $**-**Password

00000072 09 XORKey DB 9h

**;** Text **To** Show

00000073 CF 49 C1 20 3D InformationText DB "╧I┴ = ─юьiэё№ъшщ ┬рыхэЄшэ ╬ыхъёiщютшў"**,** 13**,**

20 C4 EE EC

69 ED F1 FC

EA E8 E9 20

C2 E0 EB E5

ED F2 E8 ED

20 CE EB E5

EA F1 69 E9

EE E2 E8 F7

0D C4 E0 F2

E0 20 CD E0

F0 EE E4 E6

E5 ED ED FF

20 3D 20 32

32 2E 30 32

2E 32 30 30

32 0D CD EE

EC E5 F0 20

C7 E0 EB 69

EA EE E2 EA

E8 20 3D 20

39 33 31 31

00

"─рЄр ═рЁюфцхээ  = 22.02.2002"**,** 13**,**

"═юьхЁ ╟рыiъютъш = 9311"**,** 0

000000CD CF 49 C1 20 3D InformationTextSNP DB "╧I┴ = ─юьiэё№ъшщ ┬рыхэЄшэ ╬ыхъёiщютшў"**,** 0

20 C4 EE EC

69 ED F1 FC

EA E8 E9 20

C2 E0 EB E5

ED F2 E8 ED

20 CE EB E5

EA F1 69 E9

EE E2 E8 F7

00

000000F3 C4 E0 F2 E0 20 InformationTextBirth DB "─рЄр ═рЁюфцхээ  = 22.02.2002"**,** 0

CD E0 F0 EE

E4 E6 E5 ED

ED FF 20 3D

20 32 32 2E

30 32 2E 32

30 30 32 00

00000110 0D CD EE EC E5 InformationTextZalikova DB 13**,** "═юьхЁ ╟рыiъютъш = 9311"**,** 0

F0 20 C7 E0

EB 69 EA EE

E2 EA E8 20

3D 20 39 33

31 31 00

00000128 57 69 6E 64 6F NameOfTheStartingWindows DB "Window with starting text"**,** 0 **;** the name of our window class

77 20 77 69

74 68 20 73

74 61 72 74

69 6E 67 20

74 65 78 74

00

00000142 57 69 6E 64 6F NameOfTheWarnWindows DB "Window with warn text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 77

61 72 6E 20

74 65 78 74

00

00000158 57 69 6E 64 6F NameOfFailureWindows DB "Window with failure text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 66

61 69 6C 75

72 65 20 74

65 78 74 00

00000171 57 69 6E 64 6F NameOfSuccessWindows DB "Window with some text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 73

6F 6D 65 20

74 65 78 74

00

00000187 45 64 69 74 00 NameOfTheEditBox DB "Edit"**,** 0 **;** the name of our editbox class

0000018C 42 75 74 74 6F NameOfTheButton DB "Button"**,** 0 **;** the name of our button class

6E 00

00000193 53 74 61 74 69 NameOfTheText DB "Static"**,** 0 **;** the name of our text class

63 00

0000019A CF E5 F0 E5 E2 TextForButton DB "╧хЁхт│ЁшЄш ярЁюы№"**,** 0

B3 F0 E8 F2

E8 20 EF E0

F0 EE EB FC

00

000001AC CE CA 00 TextForOKButton DB "╬╩"**,** 0

**;** Code Segment

00000000 **.**code

00000000 **start:** **;** Generates **program** start-up code

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

**invoke** GetModuleHandle**,** NULL

00000016 A3 00000000 R 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

00000030 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

00000036 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

0000003D C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

00000044 C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndProc

000005BC R

0000004B C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

00000052 C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000059 FF 35 00000000 R push hInstance

0000005F 8F 45 E4 pop wc**.**hInstance

00000062 C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**2

00000007

00000069 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

00000070 C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfTheStartingWindows

00000128 R

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

00000083 89 45 E8 mov wc**.**hIcon**,** eax

00000086 89 45 FC mov wc**.**hIconSm**,** eax

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

00000095 89 45 EC 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

000000D4 A3 00000004 R mov hWndOfMainWindow**,** eax

**;** **write** window handle in eax

000000D9 89 45 B0 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.**

00000116 8B 45 BC 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

0000011D WinMainProto endp

**;** **function** declaration of WinWarn

0000011D 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

00000123 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

0000012A C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

00000131 C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndWarnProc

00000712 R

00000138 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

0000013F C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000146 FF 35 00000000 R push hInstance

0000014C 8F 45 E4 pop wc**.**hInstance

0000014F C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000156 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

0000015D C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfTheWarnWindows

00000142 R

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

00000170 89 45 E8 mov wc**.**hIcon**,** eax

00000173 89 45 FC mov wc**.**hIconSm**,** eax

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

00000182 89 45 EC 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

000001C1 A3 00000008 R mov hWndOfWarnWindow**,** eax

**;** **write** window handle in eax

000001C6 89 45 B0 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.**

00000203 8B 45 BC 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

0000020A WinWarningProto endp

**;** **function** declaration of WinSuccess

0000020A 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

00000210 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

00000217 C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

0000021E C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndSuccessProc

000003E4 R

00000225 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

0000022C C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000233 FF 35 00000000 R push hInstance

00000239 8F 45 E4 pop wc**.**hInstance

0000023C C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000243 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

0000024A C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfSuccessWindows

00000171 R

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

0000025D 89 45 E8 mov wc**.**hIcon**,** eax

00000260 89 45 FC mov wc**.**hIconSm**,** eax

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

0000026F 89 45 EC 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

000002AE A3 0000000C R mov hWndOfSuccessWindow**,** eax

**;** **write** window handle in eax

000002B3 89 45 B0 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.**

000002F0 8B 45 BC 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

000002F7 WinSuccessProto endp

**;** **function** declaration of WinSuccess

000002F7 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

000002FD C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

00000304 C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

0000030B C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndFailureProc

00000503 R

00000312 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

00000319 C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000320 FF 35 00000000 R push hInstance

00000326 8F 45 E4 pop wc**.**hInstance

00000329 C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000330 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

00000337 C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfFailureWindows

00000158 R

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

0000034A 89 45 E8 mov wc**.**hIcon**,** eax

0000034D 89 45 FC mov wc**.**hIconSm**,** eax

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

0000035C 89 45 EC 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

0000039B A3 00000010 R mov hWndOfFailureWindow**,** eax

**;** **write** window handle in eax

000003A0 89 45 B0 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.**

000003DD 8B 45 BC 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

000003E4 WinFailureProto endp

000003E4 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

000004FD ExitCode**:**

000004FD 33 C0 xor eax**,**eax

ret

00000503 WndSuccessProc endp

00000503 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

000005B6 ExitCode**:**

000005B6 33 C0 xor eax**,** eax

ret

000005BC WndFailureProc endp

000005BC 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

0000060D A3 00000014 R 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

00000655 66**|** BB 0003 mov bx**,** 03h **;** counter for tries

00000659 81 7D 10 cmp wParam**,** 7001

00001B59

00000660 0F 85 000000A6 jne ExitCode

**;** get text from editbox

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

**;** check password's length

0000067D BF 00000000 mov edi**,** 0

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

00000682 66**|** 3D 0009 cmp ax**,** PasswordCount

**;** **...** jump **to** bad end

00000686 75 36 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

00000688 1 **??**0019**:**

00000688 47 1 inc edi

00000689 8A A7 00000018 R 1 mov ah**,** StringFromUser**[**edi**]**

0000068F 32 25 00000072 R 1 xor ah**,** XORKey

00000695 81 FF 00000009 1 cmp edi**,** PasswordCount

0000069B 75 EB 1 jne **??**0019

**;** macros #3**,** where We check Our password

IsPasswordLegit StringFromUser

0000069D 1 **??**001B**:**

0000069D 47 1 inc edi

0000069E 66**|** 3D 0009 1 cmp ax**,** PasswordCount

000006A2 74 0E 1 je **??**001A

000006A4 8A A7 00000018 R 1 mov ah**,** StringFromUser**[**edi**]**

000006AA 3A A7 00000018 R 1 cmp ah**,** StringFromUser**[**edi**]**

000006B0 74 EB 1 je **??**001B

000006B2 1 **??**001A**:**

000006B2 B9 FFFFFFF6 1 mov ecx**,** **-**10

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

000006B7 83 F9 0A cmp ecx**,** 10

000006BA 75 1D jne LegitPasswordByUser

**;** Unconditional jump **to** end

000006BC EB 00 jmp WrongPasswordByUser

000006BE WrongPasswordByUser**:**

**;** counting tries

000006BE 66**|** 83 C3 FF **add** bx**,** **-**01h **;** decrementing

000006C2 66**|** 83 FB FF cmp bx**,** **-**01h **;** negative possible tries

000006C6 74 22 je TotalExitCode

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

000006D7 EB 33 jmp ExitCode

000006D9 LegitPasswordByUser**:**

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

000006E8 EB 22 jmp ExitCode

000006EA TotalExitCode**:**

**invoke** DestroyWindow**,**hWndOfMainWindow

**.ELSE**

**;** process the message

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

ret

**.**ENDIF

0000070C ExitCode**:**

0000070C 33 C0 xor eax**,**eax

ret

00000712 WndProc endp

00000712 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

000007C5 ExitCode**:**

000007C5 33 C0 xor eax**,**eax

ret

000007CB WndWarnProc endp

end **start**

* + 1. Видно, що коментарі, які починаються з “;;” у розширеному лістингу пишуться з нового рядку, без пробілів на початку та лише з одним “;”.
    2. Будь-які коментарі не пишуться у виклику самого макросу.
    3. Можна помітити, що у виклику макросу з локальними мітками є незрозуміли символи, накшталт «??001B». це пояснюється тим, що макрогенератор змінює Наші імена на імена типу ??хххх починаючи з ??0000 включно до ??FFFF. Наприклад, в останьому виклику третього макросу є дві локальні мітки: WrongPassword та LoopItself, у розширеному лістингу вони виглядають як ??001B та ??001A відповідно.

1. Повторити дослідження, які виконані в п. 4 – 7 для другої версії програми, де всі перераховані в п. 2 і 3 макровизначення для макросів повинні розташовуватися в окремому файлі.

Сирцевий код файлу 4-9-IP93-Dominskyi-2.asm:

; 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

include 4**-**9**-**IP93**-**Dominskyi**-**2.inc

.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

Сирцевий код 4-9-IP93-Dominskyi-2.inc:

**;** 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

Лістинг файлу 4-9-IP93-Dominskyi-2.asm:

Сірий – перший макрос з виведенням тексту

Бірюзовий – другий макрос з розшифруванням паролю

Зелений – третій макрос з перевіркою паролю

include 4-9-IP93-Dominskyi-2**.**inc

C **;** Our Macroses

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

C

C **;** Macros #1 for printing some text

C PrintInformationInWindow macro heightPosition**,** infoToShow

C **;** for example**,** this commentary is included into macroexpansion

C

**;** but this **-** not

C

C **;** just pass position of the text on vertical

C **;** and text**,** that We want **to** show

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

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

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

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

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

C endm

C

C **;** Macros #2 for decrypting **string** from user

C DecryptStringFromUser macro StringFromUserInput

C **;** for example**,** this commentary is included into macroexpansion

C

**;** but this **-** not

C

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

C

**;** **when** invoking macros more then 1 time

C LOCAL LoopItself

C

C **;** creating a loop**,** **to** check all letters

C LoopItself**:**

C

C

**;** incrementing counter edi

C inc edi

C

C **;** **write** one letter from **input** **to** ah register

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

C

**;** decrypt one letter

C xor ah**,** XORKey

C

C **;** compare password **length** and counter

C cmp edi**,** PasswordCount

C

C

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

C jne LoopItself

C endm

C

C **;** Macros #3 for checking **string** from user

C IsPasswordLegit macro StringFromUserInput

C **;** for example**,** this commentary is included into macroexpansion

C

**;** but this **-** not

C

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

C

**;** **when** invoking macros more then 1 time

C LOCAL WrongPassword

C

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

C

**;** **when** invoking macros more then 1 time

C LOCAL LoopItself

C

C LoopItself**:**

C

C **;** incrementing counter edi

C inc edi

C

C

**;** compare password **length** and register

C cmp ax**,** PasswordCount

C

C **;** **if** they are the same**,** then quit macros

C je WrongPassword

C

C

**;** **write** one letter from **input** **to** ah register

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

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

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

C

C

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

C je LoopItself

C

C WrongPassword**:**

C

C **;** **set** some value**,** so our checks

C

**;** will pass**,** **IF** password is legit

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

C endm

C

00000000 **.data?**

00000000 00000000 hInstance HINSTANCE **?** **;** Handle of our **program**

00000004 00000000 hWndOfMainWindow HWND **?** **;** Handle of our main window

00000008 00000000 hWndOfWarnWindow HWND **?** **;** Handle of our warn window

0000000C 00000000 hWndOfSuccessWindow HWND **?** **;** Handle of our success window

00000010 00000000 hWndOfFailureWindow HWND **?** **;** Handle of our failure window

00000014 00000000 hWndOfEditbox HWND **?** **;** Handle of our editbox

00000018 00000080 **[** StringFromUser DB 128 dup**(?)**

00

**]**

**;** **Data** Segment

00000000 **.data**

00000000 C2 E2 E5 E4 69 StartingText DB "┬тхфiЄ№ ярЁюы№ є эрёЄєяюьє т│ъэ│, ∙юс юЄЁшьрЄш фрэ│"**,** 0

F2 FC 20 EF

E0 F0 EE EB

FC 20 F3 20

ED E0 F1 F2

F3 EF EE EC

F3 20 E2 B3

EA ED B3 2C

20 F9 EE E1

20 EE F2 F0

E8 EC E0 F2

E8 20 E4 E0

ED B3 00

00000034 CF E0 F0 EE EB FailureText DB "╧рЁюы№ эхтiЁэшщ. ╤яЁюсєщЄх ∙х Ёрч"**,** 0

FC 20 ED E5

E2 69 F0 ED

E8 E9 2E 20

D1 EF F0 EE

E1 F3 E9 F2

E5 20 F9 E5

20 F0 E0 E7

00

**;** Name Of Message Box

00000056 34 2D 39 2D 49 MsgBoxName DB "4-9-IP93-Dominskyi"**,** 0

50 39 33 2D

44 6F 6D 69

6E 73 6B 79

69 00

**;** We can **write** password in two ways**:**

00000069 4D 66 64 60 67 Password DB "Mfd`gzbp`"

7A 62 70 60

00000072

**;** And another one is**:**

**;** Password DB 31h 32h 33h

**=** 00000009 PasswordCount **=** $**-**Password

00000072 09 XORKey DB 9h

**;** Text **To** Show

00000073 CF 49 C1 20 3D InformationText DB "╧I┴ = ─юьiэё№ъшщ ┬рыхэЄшэ ╬ыхъёiщютшў"**,** 13**,**

20 C4 EE EC

69 ED F1 FC

EA E8 E9 20

C2 E0 EB E5

ED F2 E8 ED

20 CE EB E5

EA F1 69 E9

EE E2 E8 F7

0D C4 E0 F2

E0 20 CD E0

F0 EE E4 E6

E5 ED ED FF

20 3D 20 32

32 2E 30 32

2E 32 30 30

32 0D CD EE

EC E5 F0 20

C7 E0 EB 69

EA EE E2 EA

E8 20 3D 20

39 33 31 31

00

"─рЄр ═рЁюфцхээ  = 22.02.2002"**,** 13**,**

"═юьхЁ ╟рыiъютъш = 9311"**,** 0

000000CD CF 49 C1 20 3D InformationTextSNP DB "╧I┴ = ─юьiэё№ъшщ ┬рыхэЄшэ ╬ыхъёiщютшў"**,** 0

20 C4 EE EC

69 ED F1 FC

EA E8 E9 20

C2 E0 EB E5

ED F2 E8 ED

20 CE EB E5

EA F1 69 E9

EE E2 E8 F7

00

000000F3 C4 E0 F2 E0 20 InformationTextBirth DB "─рЄр ═рЁюфцхээ  = 22.02.2002"**,** 0

CD E0 F0 EE

E4 E6 E5 ED

ED FF 20 3D

20 32 32 2E

30 32 2E 32

30 30 32 00

00000110 0D CD EE EC E5 InformationTextZalikova DB 13**,** "═юьхЁ ╟рыiъютъш = 9311"**,** 0

F0 20 C7 E0

EB 69 EA EE

E2 EA E8 20

3D 20 39 33

31 31 00

00000128 57 69 6E 64 6F NameOfTheStartingWindows DB "Window with starting text"**,** 0 **;** the name of our window class

77 20 77 69

74 68 20 73

74 61 72 74

69 6E 67 20

74 65 78 74

00

00000142 57 69 6E 64 6F NameOfTheWarnWindows DB "Window with warn text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 77

61 72 6E 20

74 65 78 74

00

00000158 57 69 6E 64 6F NameOfFailureWindows DB "Window with failure text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 66

61 69 6C 75

72 65 20 74

65 78 74 00

00000171 57 69 6E 64 6F NameOfSuccessWindows DB "Window with some text"**,** 0 **;** the name of our success window class

77 20 77 69

74 68 20 73

6F 6D 65 20

74 65 78 74

00

00000187 45 64 69 74 00 NameOfTheEditBox DB "Edit"**,** 0 **;** the name of our editbox class

0000018C 42 75 74 74 6F NameOfTheButton DB "Button"**,** 0 **;** the name of our button class

6E 00

00000193 53 74 61 74 69 NameOfTheText DB "Static"**,** 0 **;** the name of our text class

63 00

0000019A CF E5 F0 E5 E2 TextForButton DB "╧хЁхт│ЁшЄш ярЁюы№"**,** 0

B3 F0 E8 F2

E8 20 EF E0

F0 EE EB FC

00

000001AC CE CA 00 TextForOKButton DB "╬╩"**,** 0

**;** Code Segment

00000000 **.**code

00000000 **start:** **;** Generates **program** start-up code

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

**invoke** GetModuleHandle**,** NULL

00000016 A3 00000000 R 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

00000030 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

00000036 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

0000003D C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

00000044 C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndProc

000005BC R

0000004B C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

00000052 C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000059 FF 35 00000000 R push hInstance

0000005F 8F 45 E4 pop wc**.**hInstance

00000062 C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**2

00000007

00000069 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

00000070 C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfTheStartingWindows

00000128 R

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

00000083 89 45 E8 mov wc**.**hIcon**,** eax

00000086 89 45 FC mov wc**.**hIconSm**,** eax

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

00000095 89 45 EC 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

000000D4 A3 00000004 R mov hWndOfMainWindow**,** eax

**;** **write** window handle in eax

000000D9 89 45 B0 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.**

00000116 8B 45 BC 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

0000011D WinMainProto endp

**;** **function** declaration of WinWarn

0000011D 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

00000123 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

0000012A C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

00000131 C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndWarnProc

00000712 R

00000138 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

0000013F C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000146 FF 35 00000000 R push hInstance

0000014C 8F 45 E4 pop wc**.**hInstance

0000014F C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000156 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

0000015D C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfTheWarnWindows

00000142 R

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

00000170 89 45 E8 mov wc**.**hIcon**,** eax

00000173 89 45 FC mov wc**.**hIconSm**,** eax

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

00000182 89 45 EC 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

000001C1 A3 00000008 R mov hWndOfWarnWindow**,** eax

**;** **write** window handle in eax

000001C6 89 45 B0 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.**

00000203 8B 45 BC 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

0000020A WinWarningProto endp

**;** **function** declaration of WinSuccess

0000020A 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

00000210 C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

00000217 C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

0000021E C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndSuccessProc

000003E4 R

00000225 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

0000022C C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000233 FF 35 00000000 R push hInstance

00000239 8F 45 E4 pop wc**.**hInstance

0000023C C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000243 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

0000024A C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfSuccessWindows

00000171 R

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

0000025D 89 45 E8 mov wc**.**hIcon**,** eax

00000260 89 45 FC mov wc**.**hIconSm**,** eax

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

0000026F 89 45 EC 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

000002AE A3 0000000C R mov hWndOfSuccessWindow**,** eax

**;** **write** window handle in eax

000002B3 89 45 B0 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.**

000002F0 8B 45 BC 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

000002F7 WinSuccessProto endp

**;** **function** declaration of WinSuccess

000002F7 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

000002FD C7 45 D0 mov wc**.**cbSize**,** sizeof WNDCLASSEX

00000030

00000304 C7 45 D4 mov wc**.**style**,** CS\_HREDRAW or CS\_VREDRAW

00000003

0000030B C7 45 D8 mov wc**.**lpfnWndProc**,** offset WndFailureProc

00000503 R

00000312 C7 45 DC mov wc**.**cbClsExtra**,** NULL

00000000

00000319 C7 45 E0 mov wc**.**cbWndExtra**,** NULL

00000000

00000320 FF 35 00000000 R push hInstance

00000326 8F 45 E4 pop wc**.**hInstance

00000329 C7 45 F0 mov wc**.**hbrBackground**,** COLOR\_WINDOW**+**1

00000006

00000330 C7 45 F4 mov wc**.**lpszMenuName**,** NULL

00000000

00000337 C7 45 F8 mov wc**.**lpszClassName**,** offset NameOfFailureWindows

00000158 R

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

0000034A 89 45 E8 mov wc**.**hIcon**,** eax

0000034D 89 45 FC mov wc**.**hIconSm**,** eax

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

0000035C 89 45 EC 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

0000039B A3 00000010 R mov hWndOfFailureWindow**,** eax

**;** **write** window handle in eax

000003A0 89 45 B0 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.**

000003DD 8B 45 BC 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

000003E4 WinFailureProto endp

000003E4 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

000004FD ExitCode**:**

000004FD 33 C0 xor eax**,**eax

ret

00000503 WndSuccessProc endp

00000503 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

000005B6 ExitCode**:**

000005B6 33 C0 xor eax**,** eax

ret

000005BC WndFailureProc endp

000005BC 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

0000060D A3 00000014 R 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

00000655 66**|** BB 0003 mov bx**,** 03h **;** counter for tries

00000659 81 7D 10 cmp wParam**,** 7001

00001B59

00000660 0F 85 000000A6 jne ExitCode

**;** get text from editbox

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

**;** check password's length

0000067D BF 00000000 mov edi**,** 0

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

00000682 66**|** 3D 0009 cmp ax**,** PasswordCount

**;** **...** jump **to** bad end

00000686 75 36 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

00000688 1 **??**0019**:**

00000688 47 1 inc edi

00000689 8A A7 00000018 R 1 mov ah**,** StringFromUser**[**edi**]**

0000068F 32 25 00000072 R 1 xor ah**,** XORKey

00000695 81 FF 00000009 1 cmp edi**,** PasswordCount

0000069B 75 EB 1 jne **??**0019

**;** macros #3**,** where We check Our password

IsPasswordLegit StringFromUser

0000069D 1 **??**001B**:**

0000069D 47 1 inc edi

0000069E 66**|** 3D 0009 1 cmp ax**,** PasswordCount

000006A2 74 0E 1 je **??**001A

000006A4 8A A7 00000018 R 1 mov ah**,** StringFromUser**[**edi**]**

000006AA 3A A7 00000018 R 1 cmp ah**,** StringFromUser**[**edi**]**

000006B0 74 EB 1 je **??**001B

000006B2 1 **??**001A**:**

000006B2 B9 FFFFFFF6 1 mov ecx**,** **-**10

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

000006B7 83 F9 0A cmp ecx**,** 10

000006BA 75 1D jne LegitPasswordByUser

**;** Unconditional jump **to** end

000006BC EB 00 jmp WrongPasswordByUser

000006BE WrongPasswordByUser**:**

**;** counting tries

000006BE 66**|** 83 C3 FF **add** bx**,** **-**01h **;** decrementing

000006C2 66**|** 83 FB FF cmp bx**,** **-**01h **;** negative possible tries

000006C6 74 22 je TotalExitCode

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

000006D7 EB 33 jmp ExitCode

000006D9 LegitPasswordByUser**:**

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

000006E8 EB 22 jmp ExitCode

000006EA TotalExitCode**:**

**invoke** DestroyWindow**,**hWndOfMainWindow

**.ELSE**

**;** process the message

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

ret

**.**ENDIF

0000070C ExitCode**:**

0000070C 33 C0 xor eax**,**eax

ret

00000712 WndProc endp

00000712 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

000007C5 ExitCode**:**

000007C5 33 C0 xor eax**,**eax

ret

000007CB WndWarnProc endp

end **start**

1. Як видно, то перед майже кожним рядком коду з файлу \*.inc стоїть літера “C”. Наскільки я зрозумів, то це позначення для стрічок, які були взяті з інших файлів.
2. Коментарі, які мають подвійний “;” у даному випадку пишуться схоже на минулий варіант, але перед ними не стоїть позначка, що вони взяті з іншого файлу.
3. Порівняти тексти розширених лістингів для обох варіантів розміщення макровизначень, знайти схожість і відмінності. Відобразити їх в звіті по лабораторній роботі

Порівнявши два лістинги можна дійти до таких висновків:

* Схоже:
  + Будь-які коментарі не пишуться у виклику самого макросу.
  + Макрогенератор змінює імена міток на імена типу ??хххх починаючи з ??0000 включно до ??FFFF
  + Видно, що коментарі, які починаються з “;;” у розширеному лістингу пишуться з нового рядку, без пробілів на початку та лише з одним “;”.
* Відмінне:
  + У другому варіанті перед майже кожним рядком стоїть позначка, яка вказує, що даний код взятий з іншого файлу, у першому ж їх немає

**Висновок:**

Я навчився краще працювати з різними віконними інтерфейсами та форматом EXE в середовищі Masm32. Ознайомився з макросами та використав їх на практиці. Також зрозумів, що певні частину коду можна писати в окремих файлах та підключати, як звичайні бібліотеки