Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра обчислювальної техніки

04-04-246М

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з навчальної дисципліни «Системне програмування» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Комп'ютерна інженерія» спеціальності 123 «Комп'ютерна інженерія» денної та заочної форм навчання

Рекомендовано науково-методичною радою з якості ННІАКОТ
Протокол № 9 від 04.07.2022 р.

Рівне – 2022

Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Системне програмування» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Комп'ютерна інженерія» спеціальності 123 «Комп'ютерна інженерія» денної та заочної форм навчання [Електронне видання] / Шатна А. В., Шатний С. В. ─ Рівне : НУВГП, 2022 ─ 69 с.

Укладачі:
Шатна А. В., старший викладач кафедри обчислювальної техніки; Шатний С. В., к.т.н., доцент кафедри обчислювальної техніки.

Відповідальний за випуск: Круліковський Б. Б., к.т.н., доцент, завідувач кафедри обчислювальної техніки.

Керівник (гарант) освітньої програми
«Комп'ютерна інженерія»
спеціальності
123 «Комп'ютерна інженерія» Сидор А. І.

Протокол № 12 засідання кафедри обчислювальної техніки від 04.06.22 р.

ЗМІСТ

Вступ

Дисципліна «Системне програмування» є складовою частиною підготовки студентів за спеціальністю 123 «Комп'ютерна інженерія». Метою викладання дисципліни «Системне програмування» є формування знань у студентів з основ системного програмування, формування навиків і вмінь в області організації та функціонування ЕОМ, структури операційних систем та їх взаємодії з прикладними програмами, набуття студентами навичок і вмінь для самостійної розробки системних програмних засобів та прикладних програм. Завданням вивчення дисципліни є ознайомлення студентів з мовою програмування API , яка максимально наближена до апаратних засобів ЕОМ, формування знань та навичок для створення компонентів операційних систем та системного програмного забезпечення. Для закріплення знань та навичків студентам необхідно виконати низку лабораторних робіт. Лабораторна робота – невеликий науковий звіт, що узагальнює проведену студентом роботу, яку представляють для захисту викладачу.

До лабораторних робіт пред’являється низка вимог, основною з яких є повний, вичерпний опис всієї проведеної роботи, що дозволяє оцінити отримані результати, міру виконання задання та професійної підготовки студентів. Звіт по лабораторній роботі друкується або пишеться студентом на одній стороні аркуша паперу формату 210х297 мм (А4).

При цьому необхідно залишати поля: зліва – 25 мм, справа – 10 мм, поверх – 20 мм, знизу – 15 мм.

Звіт повинний включати наступні пункти:

1. Мета лабораторної роботи.

2. Теоретичні відомості.

3. Індивідуальне завдання (згідно варіанту) лабораторної роботи.

4. Основні етапи та результати виконання роботи.

5. Висновки.

Лабораторна робота №1

Тема: Створення діалогових вікон засобами Windows API та їх використання у прикладних програмах.

Мета: Вивчення основних принципів створення діалогових вікон у прикладних програмах за допомогою мови сценарії ресурсів (Resource Script Language). Вивчення методів взаємодії діалогового вікна з потоком-власником та операційною системою через систему повідомлень.

Короткі теоретичні відомості

1. Створення діалогового вікна прикладної програми засобами Windows API

У методичних вказівках приведено текст програми, головне вікно якої (рис. 1.1) містить три кнопки – "Вихід" (для завершення роботи із програмою), "Діалог" (для показу діалогового вікна), та "Про вікно..." (для виводу вікна з коротким повідомленням про програму).

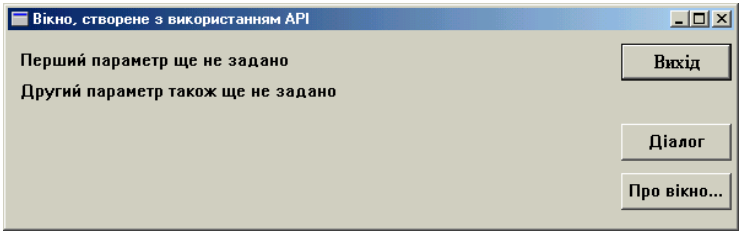

Рис. 1.1. Головне вікно програми

У вікні містяться два елементи статичного тексту (API-аналог компонента TLabel з VCL Delphi), які можуть відображувати значення двох параметрів, якщо вони будуть задані за допомогою діалогового вікна (кнопка "Діалог"). Вигляд діалогового вікна показано на рис. 1.2.

Воно дає можливість задавати два параметри- один з них вибирається з комбінованого списку (і може приймати значення "Параметр 1", "Параметр 2"...), а другий параметр є текстовим і задається з клавіатури у полі текстового вводу. Крім цього, у діалоговому вікні в нижньому лівому куті розташовано піктограму програми.

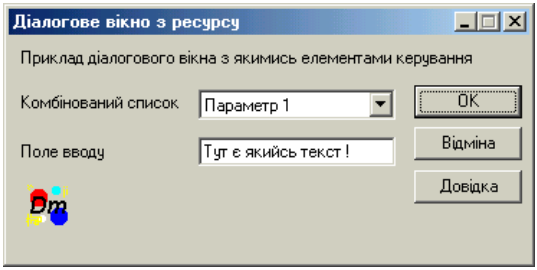

Рис. 1.2. Діалогове вікно

Для створення шаблону діалогового вікна використовують мову сценаріїв ресурсів (**Resource Script Language**), за допомогою якої описують вид вікна та інші ресурси, які у ньому використовуватимуться (малюнки, меню та інші). Так, вид сценарію для показаного на рис. 1.2 діалогового вікна наступний:

```
#define IDC_COMBOBOX1 101
#define IDC_EDIT1     102
ICIcon ICON "demo.ico"
ICDialog DIALOG 70, 50, 241, 96
STYLE DS_SYSMODAL | DS_MODALFRAME |
WS_OVERLAPPED | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX
```

```
CAPTION "Діалогове вікно з ресурсу" FONT 8, "MS Sans
Serif"
{
DEFPUSHBUTTON "OK", IDOK, 188, 21, 50, 14
PUSHBUTTON "Відміна", IDCANCEL, 188, 39, 50, 14
PUSHBUTTON "Довідка", IDHELP, 188, 57, 50, 14
COMBOBOX IDC_COMBOBOX1, 88, 23, 91, 33,
CBS_DROPDOWNLIST | WS_VSCROLL | WS_GROUP |
WS_TABSTOP
LTEXT "Приклад діалогового вікна з якимись елементами
керування", 10, 5, 5, 240, 8
EDITTEXT IDC_EDIT1, 88, 43, 91, 12
LTEXT "Комбінований список", 11, 5, 24, 79, 8
LTEXT "Поле вводу", 12, 5, 45, 60, 8
ICON "ICIcon", 20, 7, 62, 18, 20
}
```

Призначення основних елементів синтаксису мови сценаріїв ресурсів приведено у таблиці №1.1.

Таблиця №1.1

Директива, команда	**Синтаксис**	**Призначення**
#define	*#define* identifier text	При компіляції значення identifier розпізнається компілятором і автоматично заміняється на значення text Приклад: *#define IDC_COMBOBOX1 101*
ICON, оголошення типу 1	resource-name *ICO*N filename	Створює ресурс піктограми з назвою resource-name на основі заданого файлу filename Приклад: *ICIcon ICON "demo.ico"*
ICON, оголошення типу 2	*ICON* resource-name, control-ID, x, y, width, height	Розміщує у вікні піктограму, попередньо

		оголошену згідно типу 1, у заданому положенні (x та y) та заданих розмірів (width, height). Піктограмі присвоюється ідентифікатор control-ID. Приклад: *ICON "ICIcon", 20, 7, 62, 18, 20*
ICON, Оголошення типу 2	*ICON* resource-name, control-ID, x, y, width, height	Створює ресурс піктограми з назвою resource-name на основі заданого файлу filename Приклад: *ICIcon ICON "demo.ico"*
DIALOG	resource-name *DIALOG* x, y, width, height *[STYLE* w-style] *[CAPTION* w-cap] *[MENU* res-name] *[CLASS* w-class] [FONT f-spec] *BEGIN* dialog-controls *END*	Розміщує у вікні піктограму, попередньо оголошену згідно типу 1, у заданому положенні (x та y) та заданих розмірів (width, height). Піктограмі присвоюється ідентифікатор control-ID. Приклад: *ICON "ICIcon", 20, 7, 62, 18, 20*
STYLE	*STYLE* w-style	Описує діалогове вікно. Може включати стиль вікна, його клас, розмір (width, height), положення (x та y), та елементи керування (які описуються в межах конструкції *BEGIN... END [або {... }]).*
CAPTION	*CAPTION* w-cap	Задає заголовок w-cap для діалогового вікна.
Директива, команда	**Синтаксис**	**Призначення**
MENU	*MENU* res-name	Задає ресурс меню діалогового вікна, який повинен бути попередньо описаним.
CLASS	*CLASS* w-class	Задає клас діалогового вікна. Якщо конструкція відсутня, створюється стандартне вікно.

FONT	*FONT* f-spec	Задає шрифт для діалогового вікна. f-spec задає розмір та назву шрифта. Приклад: *FONT 8, "MS Sans Serif"*

BEGIN END
або
{
}
		Оператори задають початок та кінець єдиної багатолінійної конструкції.

DEFPUSHBUTTON	*DEFPUSHBUTTON* text, control-ID, x, y, width, height, [c-style]	Створює кнопку по замовчуванню у діалоговому вікні. Кнопка містить напис text, ідентифікатор ресурсу control- ID, екранні координати x, y, та розміри width, height. Кнопка може містити стилі: *WS_DISABLED* - елемент керування неактивний *WS_GROUP* - елемент згрупований *WS_TABSTOP* - при навігації по елементам керування вікна за допомогою клавіші Tab здійснюватиметься зупинка на даній кнопці. Приклад: *DEFPUSHBUTTON "OK", IDOK, 188, 21, 50, 14*
PUSHBUTTON	*PUSHBUTTON* text, control-ID, x, y, width, height, [c-style]	Створює кнопку у діалоговому вікні. Кнопка містить напис text, ідентифікатор ресурсу *control-ID,* екранні координати x, y, та розміри width, height. Кнопка може містити стилі [c-style]. Приклад: *PUSHBUTTON "Відміна", IDCANCEL, 188, 39, 50, 14*
COMBOBOX	*COMBOBOX ID*, x, y, width, height, [c-style]	Створює комбінований список у діалоговому вікні. Він містить ідентифікатор ресурсу ID,

		екранні координати x, y, розміри width, height, та може містити стилі [c-style]. Приклад: *COMBOBOX 2, 88, 23, 91, 33, CBS_DROPDOWNLIST | WS_VSCROLL | WS_GROUP | WS_TABSTOP*
LTEXT	*LTEXT* text, control-ID, x, y, width, height, [c-style]	Створює вирівняний по лівому краю текст text у діалоговому вікні. Він містить ідентифікатор ресурсу control-ID, екранні координати x, y, розміри width, height, та може містити стилі [c-style]. *Приклад:* *LTEXT "Текст", 12, 5, 45, 60, 8*
EDITTEXT	*EDITTEXT* control-ID, x, y, width, height, [c-style]	Створює поле тектового вводу (поле редагування) у діалоговому вікні. Воно містить ідентифікатор ресурсу control-ID, екранні координати x, y, розміри width, height, та може містити стилі [c-style]. *Приклад:* *EDITTEXT 1, 88, 43, 91, 12*

Файл сценарію зберігається як звичайний текстовий файл 8, і компілюється, наприклад, з використанням компілятора ресурсів фірми Borland BRCC32.EXE, який поставляється разом з Delphi9. Якщо файл сценарію назвати ICDIALOG.RC, то в результаті його компіляції отримається файл ресурсів ICDIALOG.RES, який міститиме шаблон розробленого діалогу. Для під′єднання цього ресурсного файлу до проекту Delphi використовується така директива компілятора 10:

{$R ICDIALOG.RES}

Якщо шаблон діалогового вікна вже створено та відповідний ресурсний файл під’єднано до проекту, то само вікно можна створити за допомогою функції DialogBox 11:

DialogBox(hInstance,'ICDialog',hWindow,@DlgProc)

Ця функція як параметри отримує ідентифікатор екземпляра запущеної програми hInstance, назву ресурсу діалогового вікна (в даному випадку 'ICDialog' – див. описаний вище сценарій діалогового вікна), ідентифікатор вікна-власника для створюваного діалогу (hWindow), та адресу

8 Для цього можна використати стандартний текстовий редактор "Блокнот" (Notepad) Windows.

9 При цьому назва файлу сценарію, який слід скомпілювати, вказується як перший параметр командного рядка компілятора. Так, якщо файл сценарію має назву ICDIALOG.RC, то командний рядок має виглядати так:

BRCC32.EXE ICDIALOG.RC

10 Її можна помістити відразу після директиви, яку Delphi додає у проект автоматично – {$R *.res}, і яка під′єднує файл ресурсів самого проекту (його назва співпадає з назвою проекту).

11 Опис функцій Windows API приведено у додатку. Для повного опису функцій Windows API див. довідку MS SDK. точки входу віконної функції діалогового вікна (@DlgProc 12). Як результат функція *DialogBox* повертає ідентифікатор натисненої кнопки.

Нам необхідно, щоб при натисканні кнопки "ОК" у діалоговому вікні задані у ньому два параметри зберігалися у глобальних змінних. В описуваній програмі такими змінними є: ParamEdit – для зберігання тексту з поля редагування, та ParamCB – для зберігання тексту з

комбінованого списку. У Windows API для зберігання текстової інформації використовуються рядки з кінцевим нулем (у Object Pascal стандартним засобом для цього є тип string). Рядок з кінцевим нульовим символом в Object Pascal оголошується як тип PChar, що представляє собою вказівник 13 на буфер, у якому розміщено рядок. Оскільки рядок представляє собою масив символів (який закінчується символом з кодом 0), то його можна й оголосити як масив (array).

Для ілюстрації різних методів роботи з рядками, які містять кінцевий нуль, у даній програмі змінні двох параметрів оголошені по-різному:

ParamEdit : array [1..50] of char;
ParamCB : PChar;

При оголошенні змінної як масив у сегменті даних програми резервується відповідне місце для елементів цього масиву, що приводить до збільшення розмірів виконавчого файлу на розмір масиву. Якщо змінна оголошується як тип PChar (тобто вказівник, який без попередньої ініціалізації може вказувати на довільну ділянку пам′яті), то перед використанням відповідної змінної слід виділити небхідний обсяг пам’яті для рядка:

GetMem(ParamCB,50);

В даномі випадку виділяється 50 байт, що не дозволить обробляти рядки довжиною більше 49 символів 14. Так як у програмі не здійснюється обмеження по довжині введеного тексту та відповідна перевірка, то при спробі роботи з довшими рядками може виникати помилка доступу до пам’яті. Після того, як буфер для рядка вже непотрібний, слід вивільнити попередньо зарезервовану пам’ять:

FreeMem(ParamCB);

Розглянемо віконну функцію для створеного діалогового вікна:

12 Оператор @ вказує на адресу програмного об'єкта (змінної, функції, процедури...), перед яким стоїть.
13 Вказівник – це змінна, яка містить адресу програмного об'єкта (тобто іншої змінної, масиву, структури, процедури, функції і т.д.)
14 Один байт має залишатись для кінцевого символа з кодом 0.

```
function DlgProc(Window : hWnd; Msg,WParam,LParam : Integer): Integer; stdcall;
begin
Result:=0; case Msg of
WM_INITDIALOG : begin
// Ініціалізація діалогового вікна Result:=0; hcbType:=GetDlgItem(Window,101);
    SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));
    SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 2')));
    SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 3')));
    SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 4')));
    SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр5')));
SendMessage(hcbType,CB_SETCURSEL,0,0);

      hEdit:=GetDlgItem(Window,102);
SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Тут є якийсь текст !')));
      end;
      WM_COMMAND    : if (LoWord(WParam)=IDOK) then begin // Натиснено
кнопку "OK" SendMessage(hEdit,WM_GETTEXT,
        SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));
         SendMessage(hcbType,WM_GETTEXT,
SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));
            EndDialog(Window,idOK)
      end          // Натиснено кнопку "Відміна"
      else if (LoWord(WParam)=IDCANCEL) then EndDialog(Window,idCancel) else   //
Натиснено кнопку "Довідка"
           if (LoWord(WParam)=IDHELP) then MessageBox(Window,
      'Довідка про діалогове вікно, яке створене з використанням '+ 'Microsoft®
Windows® Application Program Interface®.'+ #13#13+'Copyright© Microsoft® & K°',
      'Довідка про API Window',MB_OK or
MB_ICONINFORMATION); WM_CLOSE:
EndDialog(Window,idCancel);
```

Повідомлення WM_INITDIALOG використовується для ініціалізації новоствореного діалогового вікна, оскільки воно отримується віконною функцією один раз безпосередньо після його створення. Так як для роботи з будь-яким вікном (в тому числі з вікном елемента керування) потрібно знати його ідентифікатор, то спочатку отримуєм ідентифікатор комбінованого списку:

hcbType:=GetDlgItem(Window,101);

Функція GetDlgItem повертає ідентифікатор вікна елемента керування діалогового вікна за його номером. У даному випадку визначається ідентифікатор комбінованого списку (його номер у шаблоні діалогового вікна – 101 (див. приведений вище текст сценарію діалогового вікна)). Після цього у комбінований список добавляються рядки з необхідними значеннями. Це здійснюється шляхом посилання його вікну повідомлення CB_ADDSTRING:

SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));

Другий параметр цього повідомлення містить адресу рядка 'Параметр 1', який і добавиться у список. Таким чином у список поміщуються п’ять рядків з різними значеннями. Після цього визначається ідентифікатор поля текстового вводу hEdit:

hEdit:=GetDlgItem(Window,102);

За допомогою повідомлення WM_SETTEXT у це поле вноситься текст ('Тут є якийсь текст !'):

SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Ty т є якийсь текст !')));

На цьому ініціалізація вікна закінчується.

Повідомлення WM_COMMAND в нашому випадку отримується при натисканні на кнопки діалогового вікна. При цьому нижнє слово параметра wParam містить ідентифікатор ресурсу елемента керування, від якого отримано повідомлення (див. текст сценарію діалогового вікна). Так, наприклад, для кнопки "ОК" для описуваного діалогу цей ідентифікатор дорівнює IDOK. Якщо у діалоговому вікні було натиснено кнопку "ОК", то текст з комбінованого списку та поля редагування зчитується і запам’ятовується у відповідних змінних.

Для отримання тексту з відповідного текстового вікна йому посилається повідомлення WM_GETTEXT. При цьому перший параметр повідомлення має вказувати кількість символів, які будуть отримані з вікна, а другий параметр – на буфер, що отримає значення рядка. Для обчислення довжини рядка у вікні йому посилається повідомлення WM_GETTEXTLENGTH, яке результатом повертає кількість символів у рядку вікна. Так як при цьому не враховується останній, нульовий символ, то для задання розміру тексту до результату повідомлення WM_GETTEXTLENGTH додається 1, і результат цієї операції передається як перший параметр повідомлення WM_GETTEXT. Другий параметр цього повідомлення повинен містити адресу рядка, який прийме текстове значення. При цьому для змінної ParamEdit, яка оголошена як масив символів, визначається адреса першого елемента масиву (@ParamEdit) та приводиться до типу DWORD, щоб бути сумісним з відповідним параметром повідомлення:

SendMessage(hEdit,WM_GETTEXT,SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));

Для змінної ParamCB (типу PChar), яка сама є вказівником на рядок, адресу не потрібно обчислювати:

SendMessage(hcbType,WM_GETTEXT,SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));

Після отримання значень, заданих у діалоговому вікні, воно знищується за допомогою функції EndDialog, яка повертає значення IDOK.

Якщо у діалоговому вікні було натиснено кнопку "Відміна", воно знищується, повертаючи результат IDCANCEL. При виборі кнопки "Довідка" з'являється вікно з повідомленням про дане діалогове вікно.

Останнє повідомлення, яке обробляється віконною функцією діалогового вікна- WM_CLOSE.

При цьому воно також знищується.

Якщо у діалоговому вікні натискається кнопка "ОК", викликається процедура ChangeParams (див. віконну функцію головного вікна WindowProc), яка відображає вибрані в діалозі параметри у головному вікні програми за допомогою елементів статичного тексту:

procedure ChangeParams; begin

SendMessage(hText1,WM_SETTEXT,0,DWORD('Вибраний параметр з комбінованого списку: '+ParamCB)); SendMessage(hText2,WM_SETTEXT,0,DWORD(@ParamEdit));

end;

Запис тексту у відповідний елемент здійснюється шляхом посилання йому повідомлення WM_SETTEXT, другий параметр якого містить адресу буфера, що містить рядок для запису у вікно. Таким чином введені у діалоговому вікні параметри стають доступними в головному вікні програми.

2. Завдання на лабораторну роботу

Написати (з використанням лише функцій Windows API, без впровадження компонентів VCL Delphi) програму, яка здійснюватиме розрахунок арифметичного виразу згідно

варіанту (таблиця №1.2) та відображуватиме його в головному вікні у полі редагування (EDIT). Значення змінних, які використовуються для розрахунку, повинні задаватися за допомогою окремого діалогового вікна. Програма повинна здійснювати перевірку можливості обчислення виразу при заданих параметрах та у випадку неможливості проведення обчислень видавати відповідне повідомлення 15.

Головне вікно має містити кнопку "Вихід", за якою здійснюватиметься коректне завершення роботи програми.

Таблиця 1.3.

Варіант	**Завдання**
1	$y = \cos(ab)\sqrt{ab}$
2	$y = \dfrac{\sqrt{a - b^3}}{\sin(ab + \frac{a}{b})}$
3	$y = cos^3(a) + sin^2(b)$
4	$y = \dfrac{\ln(a)}{a + 4.23b + ab}$
5	$y = (356a - 23b)(\sqrt{3a - b})$
6	$y = 2.36\pi + 2.58cos^5(3ab)$
7	$y = 2a + \dfrac{3b^3 - 6a}{6a^2 - b}$
8	$y = 78a^6 + 32b^4 - 23ab^3$
9	$y = \dfrac{-\lvert a^3 + 4b^5 \rvert}{\sqrt[3]{3ab}}$
10	$y = \dfrac{\sqrt[4]{a + 3b}}{(2a + b)(3b - 2a)}$

Текст програми
program Window;

```
uses Windows, Messages, SysUtils;

const AppName='API Window';

        var AMessage : TMsg;
        hWindow : HWnd;
        hBtnExit : HWnd;
        hBtnAbout: HWnd;
        hBtnDlg   : HWnd;
        hText1      : HWnd;
        hText2      : HWnd;

var hcbType : hWnd;
hEdit: hWnd;

ParamEdit : array [1..50] of char;
ParamCB           : PChar;
{$R ICDIALOG.RES}

function DlgProc(Window : hWnd; Msg,WParam,LParam : Integer): Integer; stdcall;
begin
  Result:=0;
  case Msg of
WM_INITDIALOG : begin
// Ініціалізація діалогового вікна
        Result:=0;
        hcbType:=GetDlgItem(Window,101);
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 1')));
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 2')));
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 3')));
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр 4')));
        SendMessage(hcbType,CB_ADDSTRING,0,DWORD(PChar('Параметр5')));
SendMessage(hcbType,CB_SETCURSEL,0,0);

hEdit:=GetDlgItem(Window,102);
SendMessage(hEdit,WM_SETTEXT,0,DWORD(PChar('Тут є якийсь текст !')));
end;
```

WM_COMMAND: if (LoWord(WParam)=IDOK) then begin *// **Натиснено кнопку "ОК"***

```
      SendMessage(hEdit,WM_GETTEXT,
```

```
      SendMessage(hEdit,WM_GETTEXTLENGTH,0,0)+1,DWORD(@ParamEdit));
SendMessage(hcbType,WM_GETTEXT,
      SendMessage(hcbType,WM_GETTEXTLENGTH,0,0)+1,DWORD(ParamCB));
EndDialog(Window,idOK)
      end// Натиснено кнопку "Відміна"

else if (LoWord(WParam)=IDCANCEL) then EndDialog(Window,idCancel) else
          // Натиснено кнопку "Довідка"
if (LoWord(WParam)=IDHELP) then MessageBox(Window,
'Довідка про діалогове вікно, яке створене з використанням Microsoft® '+
'Windows® Application Program Interface®.'+ #13#13+'Copyright© Microsoft® &
K°', 'Довідка про API Window',MB_OK or MB_ICONINFORMATION);

WM_CLOSE     : EndDialog(Window,idCancel);
      else Result:=0;
         end;
      end;

procedure ChangeParams;
begin
    SendMessage(hText1,WM_SETTEXT,0,DWORD('Вибраний параметр з
комбінованого списку: '+ParamCB));
SendMessage(hText2,WM_SETTEXT,0,DWORD(@ParamEdit));
end;

function WindowProc(Window:HWnd; AMessage, WParam,
LParam:longint):longint;stdcall;
    export;
begin
    WindowProc:=0;
    case AMessage of
WM_DESTROY: begin PostQuitMessage(0); Exit;
end;

WM_COMMAND: if HWnd(LParam)=hBtnExit then begin PostQuitMessage(0); Exit;

    end else if HWnd(LParam)=hBtnAbout then MessageBox(Window,
    'Вікно, створене функціями API, без використання VCL Delphi.'+#13#13+
'Copyright© Microsoft... і т.д.','API Window',MB_OK or MB_ICONINFORMATION)
    else if HWnd(LParam)=hBtnDlg then begin
```

```
    if DialogBox(hInstance,'ICDialog',hWindow,@DlgProc)=IDOK then
ChangeParams; end;

WM_KEYDOWN: if (WParam=VK_RETURN) or (WParam=VK_ESCAPE) then
SendMessage(hBtnExit,BM_CLICK,0,0)
    else if WParam=VK_F1 then SendMessage(hBtnAbout,BM_CLICK,0,0);
end;

WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;

function WinRegister: boolean;
var WindowClass: TWndClass;
begin
  with WindowClass do begin Style:=CS_HREDRAW or CS_VREDRAW;
            lpfnWndProc:=@WindowProc;
            cbClsExtra:=0;
            cbWndExtra:=0;
            hInstance:=HInstance;
            hIcon:=LoadIcon(0, IDI_APPLICATION);
            hCursor:=LoadCursor(0, IDC_ARROW);
            hbrBackGround:=COLOR_WINDOW;
            lpszMenuName:=nil;
            lpszClassName:=AppName;
            end;
            Result:=RegisterClass(WindowClass)<>0;
            end;

function WinCreate: HWnd;
var hWindow : HWnd;
begin
  hWindow:=CreateWindow(AppName, 'Вікно, створене з використанням
API',WS_OVERLAPPEDWINDOW, 200, 200, 600, 185, 0, 0, HInstance, nil);

  if hWindow<>0 then
  begin
  ShowWindow(hWindow, SW_SHOWNORMAL);
   UpdateWindow(hWindow);
  end;

hBtnExit:=CreateWindow('BUTTON','Вихід',WS_CHILD or BS_DEFPUSHBUTTON
or WS_TABSTOP, 500, 10, 90, 30, hWindow, 0, HInstance, nil);
```

```
hBtnDlg:=CreateWindow('BUTTON', 'Діалог',WS_CHILD or WS_TABSTOP, 500, 75,
90, 30, hWindow, 0, HInstance, nil);

if hBtnDlg<>0 then ShowWindow(hBtnDlg, SW_SHOWNORMAL);
hBtnAbout:=CreateWindow('BUTTON', 'Про вікно...',WS_CHILD or WS_TABSTOP,
500, 115, 90, 30, hWindow, 0, HInstance, nil);
if hBtnAbout<>0 then ShowWindow(hBtnAbout, SW_SHOWNORMAL);
  hText1:=CreateWindow('STATIC', 'Перший параметр ще не задано',WS_CHILD,
10, 15, 450, 20, hWindow, 0, HInstance, nil);
if hText1<>0 then ShowWindow(hText1, SW_SHOWNORMAL);
  hText2:=CreateWindow('STATIC', 'Другий параметр також ще не
задано',WS_CHILD, 10, 40, 450, 20, hWindow, 0, HInstance, nil);
if hText2<>0 then ShowWindow(hText2, SW_SHOWNORMAL); Result:=hWindow;
end;
beginif not WinRegister then begin
MessageBox(0,   // обробник батьківського вікна
          'Клас вікна не зареєстровано', // адреса тексту повідомлення
          'API Window',    // адреса заголовку вікна повідомлення
          MB_OK);          // стиль вікна повідомлення
Exit;
end;

hWindow:=WinCreate;
if hWindow=0 then
begin
    MessageBox(0, 'Не вийшло створити вікно.', 'API Window', MB_OK);
    Exit;
end;

GetMem(ParamCB,50);
    while GetMessage(AMessage,0,0,0) do begin
    TranslateMessage(AMessage); DispatchMessage(AMessage);
end;

    FreeMem(ParamCB);
    Halt(AMessage.wParam);
end
```

Завдання для виконання роботи

Написати програму для одержання відомостей про систему, використавши функції Win32 API типу:

GetSystemDirectory(), GetWindowsDirectory(), GetComputerName(), GetUserName(), GetVersionEx(), GetKeyboardType(). Для виведення результатів використати функції wsprintf(), WriteConsole().

Написати програму для одержання відомостей про систему, використавши функції Win32 API: GetSystemInfo(), GetSysColor(), GetSystemMetrics(). Передбачити введення команд з клавіатури.

Написати програму, яка виводить в задану позицію консолі інформацію про координати миші, стан керуючих клавіш (Alt, Ctrl, Shift), скан-код та ASCII-код натиснутої клавіші. Для виведення результатів використати функції SetConsoleCursorPosition(), wsprintf(), WriteConsole(),

Контрольні питання

Що таке консольна Windows-програма?
Які стандартні пристрої введення-виведення використовує консоль?
Що таке вхідний буфер та екранний буфер консолі?
Що таке високорівневий та низькорівнеаий доступ до буферів консолі?
Які високорівневі консольні функції Win32 API ви знаєте?
Які можливості надають низькорівневі консольні функції?
Як визначити стан кнопок миші?
Як визначити стан керуючих клавіш клавіатури?

Лабораторна робота № 2

Тема: Консольні Win32-програми для Windows.

Мета: Ознайомитися із принципами і засобами створення консольних програм для Windows. Навчитися використовувати функції Win32 API для роботи з консоллю.

Короткі теоретичні відомості

Створення консолі

Microsoft Windows створює нову консоль, коли вона запускає процес символьного режиму. Наприклад, Windows створює нову консоль, коли вона запускає командний процесор cmd.exe. Коли командний процесор запускає новий процес консолі, користувач може визначити чи система створює нову консоль для нового процесу чи вона успадковує консоль командного процесора.

Якщо запустити консольний додаток із командного рядка, то додаток використовує "чужу" консоль. Для створення своєї консолі потрібно від'єднатися від чужої консолі і створити свою. Для цього використовуються наступні функції:

BOOL FreeConsole(VOID) - *від'єднує процес від консолі.*

BOOL AllocConsole(VOID) - *виділяє нову консоль для процесу.*

Для роботи із консоллю потрібно одержати дескриптор консолі:

HANDLE GetStdHandle(DWORD nStdHandle);

де nStdHandle визначає пристрій, для якого одержують дескриптор:

STD_INPUT_HANDLE (-10) Стандартний пристрій вводу
STD_OUTPUT_HANDLE (-11) Стандартний пристрій виводу

STD_ERROR_HANDLE (-12) Стандартний пристрій помилок

Наприклад:

hStdIn = GetStdHandle(STD_INPUT_HANDLE);
hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

Після цього усі операції із консоллю здійснюються через її дескриптор. Якщо функція завершується фатально, то вона повертає значення INVALID_HANDLE_VALUE.

Налаштування консолі

Для настройки параметрів консолі використовують такі функції:

Функція API	**Коментарі**
DWORD **GetConsoleTitle(** LPTSTR **lpConsoleTitle**, DWORD **nSize);**	*Одержати заголовок вікна консолі* адреса буферарозмір буфера
BOOL **SetConsoleTitle(** LPCTSTR **lpConsoleTitle);**	*Встановити заголовок вікна консолі* адреса нового заголовка
BOOL **SetConsoleMode(** HANDLE **hConsoleHandle,** DWORD **dwMode);**	*Встановити режим консолі* дескриптор вхідного або екранного буфера вхідний або вихідний режим
BOOL **SetConsoleCursorPosition(** HANDLE **hConsoleOutput,** COORD **dwCursorPosition);**	*Встановити нові координати курсору* дескриптор екранного буфера нові координати курсору
BOOL **SetConsoleTextAttribute(** HANDLE **hConsoleOutput,** WORD **wAttributes);**	*Встановити атрибути тексту консолі* дескриптор екранного буфера кольори тексту і фону

Закрити консоль можна за допомогою функції **FreeConsole.** Якщо інші процеси підключені до консолі, вона зберігається. Коли останній процес від'єднується від консолі, вона закривається.

Функції для роботи з консоллю

Функції Win32 API допускають два рівні доступу до консолі: високорівневий і низькорівневий.

Високорівневі вхідні функції фільтрують вміст вхідного буфера і повертають лише потік символів з клавіатури, відкидаючи службову інформацію та інші події.

Високорівневі функції виведення записують потік символів у буфер екрану, які відображаються в поточному розташуванні курсора.

Високорівневі функції також підтримують перепризначення стандартних дескрипторів і керують режимами консолі.

Повний перелік функцій для роботи з консоллю можна знайти в довідковій системі Microsoft Win32 Programmer's Reference у розділі Consoles and Character-Mode Support Console Functions.

Високорівневі функції

Програма може використовувати файлові функції вводу/виводу ReadFile і WriteFile, ф також спеціалізовані консольні функції ReadConsole і WriteConsole, для високорівневого I/O, який надає непрямий доступ до вхідного і екранного буферів.

BOOL **ReadConsole(**	
HANDLE **hConsoleInput,**	*// дескриптор вхідного буфера консолі*
LPVOID **lpBuffer,**	*// адреса буфера прийому даних*
DWORD **nNumberOfCharsToRead,**	*// кількість символів для читання*
LPDWORD **lpNumberOfCharsRead,**	*// адреса числа прочитаних символів*
LPVOID **lpReserved);**	*// зарезервовано, завжди NULL*
BOOL **WriteConsole(**	*// дескриптор екранного буфера консолі*
HANDLE **hConsoleOutput,**	*// адреса буфера даних для виводу*
CONST VOID ***lpBuffer,**	*// кількість символів для запису*
DWORD **nNumberOfCharsToWrite,**	*// вказівник числа прочитаних символів*

LPDWORD **lpNumberOfCharsWritten,** LPVOID **lpReserved);**	*// зарезервовано, завжди NULL*

Функції **ReadConsole, WriteConsole** оперують з текстовими рядками. При підготовці текстових рядків для цих функцій корисною може бути функція Win API wsprintf.

Ця функція призначена для форматованого виведення послідовності символів і значень у текстовий буфер. Дія функції схожа на функцію printf, але вона виводить не на екран, а в текстовий буфер. Функція автоматично додає кінцевий нуль, хоча він не входить у число виведених символів, яке повертає функція по завершенні своєї роботи.

```
int wsprintf(
  LPTSTR lpOut,        // pointer to buffer for output
  LPCTSTR lpFmt,       // pointer to format-control string
  ...                  // optional arguments
);
```

Параметри

lpOut вказівник на буфер для прийому символів.

lpFmt вказівник на рядок форматування із кінцевим нулем.

Після цього йдуть значення, тип і кількість яких відповідає рядку форматування.

Функція повертає кількість виведених у буфер символів (за винятком кінцевого

нуля).

Низькорівневі функції

Низькорівневі функції вводу забезпечують прямий доступ до вхідного буфера і дають можливість додаткам отримувати детальні дані про події клавіатури і миші, а також про події, що визначають взаємодію користувача із вікном консолі.

Низькорівневі функції виводу дають можливість додатку читати або записувати певну послідовність комірок символів у буфер екрану.

Високорівневі і низькорівневі методи I/O не взаємовиключні, і програма може використовувати будь-яку комбінацію цих функцій. Проте, зазвичай прикладні програми використовують або один підхід або інший.

Низькорівневі функції вводу

Win32 API надає п'ять низькорівневих функцій для доступу до вхідного буфера консолі:

ReadConsoleInput	Читає і видаляє записи із вхідного буфера. Функція не повертається, поки є хоча б один запис, доступний для читання. Прочитані записи переміщаються в буфер викликаючого процесу. Непрочитані записи залишаються у вхідному буфері для наступної операції читання даних. Функція повідомляє загальне число записів, які були прочитані.
PeekConsoleInput	Читає вхідні записи без видалення із вхідного буфера. Якщо жоден запис недоступний, функція повертається негайно. Функція повідомляє загальне число записів, які були прочитані.
GetNumberOfConsole-InputEvents	Визначає число непрочитаних вхідних записів у вхідному буфері.
WriteConsoleInput	Розміщує записи у вхідному буфері позаду наявних записів. Вхідний буфер зростає динамічно, якщо необхідно утримувати додаткові вхідні записи.
FlushConsoleInputBuffer	Відкидає всі непрочитані події у вхідному буфері (очищає буфер).

Низькорівневі функції виводу

Низькорівневі функції виводу консолі надають прямий доступ до символьних комірок буфера екрану. Одні функції

читають або записують до послідовних комірок екранного буфера, починаючи із будь-якої позиції. Інші функції читають або записують до прямокутних блоків комірок.

Наступні функції читають або записують до вказаної послідовності символьних комірок в буфері екрану, починаючи із вказаної.

ReadConsoleOutputCharacter	Копіює рядок символів Unicode або ANSI із буфера екрану.
WriteConsoleOutputCharacter	Записує рядок символів Unicode або ANSI до буфера екрану.
ReadConsoleOutputAttribute	Копіює атрибути кольору рядка тексту і фону з буфера екрану.
WriteConsoleOutputAttribute	Записує атрибути кольору рядка тексту і фону до буферу екрану.
FillConsoleOutputCharacter	Записує єдиний Unicode або символ ANSI до вказаної кількості послідовних комірок в буфері екрану.
FillConsoleOutputAttribute	Записує атрибути кольору рядка тексту і фону до вказаної кількості послідовних комірок в буфері екрану.

Низькорівневі функції вимагають писати більше коду і вибирати серед більшого діапазону функцій, проте він також надає програмі більшої гнучкості.

Для роботи із вхідним буфером консолі за допомогою низькорівневих функцій слід ознайомитися із структурою запису у вхідному буфері. Кожний запис у вхідному буфері складається із типу події (WORD EventType) і запису про подію. Тип події приймає наступні значення: KEY_EVENT, MOUSE_EVENT, WINDOW_BUFFER_SIZE_EVENT, MENU_EVENT, FOCUS_EVENT (подія клавіатури, миші, зміни розміру буфера, меню, фокусу).

Розглянемо для прикладу структуру записів про подію від клавіатури і миші. Вони мають такі прототипи:

Подія клавіатури	Подія миші
typedef struct _KEY_EVENT_RECORD { BOOL bKeyDown; WORD wRepeatCount; WORD wVirtualKeyCode; WORD wVirtualScanCode; union { WCHAR UnicodeChar; CHAR AsciiChar; } uChar; DWORD dwControlKeyState; } KEY_EVENT_RECORD;	typedef struct _MOUSE_EVENT_RECORD { COORD dwMousePosition; DWORD dwButtonState; DWORD dwControlKeyState; DWORD dwEventFlags; } MOUSE_EVENT_RECORD;

Повний запис у вхідному буфері для подій клавіатури і миші має такий вигляд:

EventType	Event						
MOUSE EVENT	MOUSE_EVENT_RECORD						
MOUSE_-EVENT	dwMouse-Position		dwButtonState		dwControlKeyState		dwEventFlags
	wX	wY					
0	4	6	8		12		16
KEY EVENT	KEY_EVENT_RECORD						
KEY_EVENT	bKeyDown		wRepeat Count	wVirtual-KeyCode	wVirtual-ScanCode	uChar	dwControl-KeyState
0	4		8	10	12	14	16

Записи у вхідному буфері консолі містять поля, які відображають стан кнопок миші (dwButtonState), стан керуючих клавіш клавіатури (dwControlKeyState) та тип події миші (dwEventFlags). Ці поля можуть приймати наступні значення.

dwButtonState:	**dwControlKeyState:**
FROM_LEFT_1ST_BUTTON_PRESSED	RIGHT_ALT_PRESSED
RIGHTMOST_BUTTON_PRESSED	LEFT_ALT_PRESSED
FROM_LEFT_2ND_BUTTON_PRESSED	RIGHT_CTRL_PRESSED
FROM_LEFT_3RD_BUTTON_PRESSED	LEFT_CTRL_PRESSED
FROM_LEFT_4TH_BUTTON_PRESSED	SHIFT_PRESSED

dwEventFlags: DOUBLE_CLICK MOUSE_MOVED MOUSE_WHEELED	NUMLOCK_ON SCROLLLOCK_ON CAPSLOCK_ON ENHANCED_KEY

Розглянемо приклади низькорівневих функцій:

BOOL ReadConsoleInput(	
HANDLE hConsoleInput,	// дескриптор вхідного буфера консолі
PINPUT_RECORD lpBuffer,	// адреса буфера для читання даних
DWORD nLength,	// кількість записів, які слід прочитати
LPDWORD lpNumberOfEventsRead	// адреса числа прочитаних записів
);	

Функція запису в екранний буфер консолі:

BOOL WriteConsoleOutputCharacter(	
HANDLE hConsoleOutput,	// дескриптор екранного буфера консолі
LPCTSTR lpCharacter,	// вказівник на буфер, із якого виводять
DWORD nLength,	// кількість комірок для запису
COORD dwWriteCoord,	// координати першої комірки
LPDWORD lpNumberOfCharsWritten	//вказівник на число записаних комірок
);	

Функція установки координат курсору:

BOOL SetConsoleCursorPosition(	
HANDLE hConsoleOutput,	// дескриптор екранного буфера консолі
COORD dwCursorPosition	// нові координати курсору
);	

Завдання для виконання роботи

1. Написати програму для одержання відомостей про систему, використавши функції Win32 API типу: GetSystemDirectory(), GetWindowsDirectory(),

GetComputerName(), GetUserName(), GetVersionEx(), GetKeyboardType(). Для виведення результатів використати функції wsprintf(), WriteConsole().

2. Написати програму для одержання відомостей про систему, використавши функції Win32 API: GetSystemInfo(), GetSysColor(), GetSystemMetrics(). Передбачити введення команд з клавіатури.

3. Написати програму, яка виводить в задану позицію консолі інформацію про координати миші, стан керуючих клавіш (Alt, Ctrl, Shift), скан-код та ASCII-код натиснутої клавіші. Для виведення результатів використати функції SetConsoleCursorPosition(), wsprintf(), WriteConsole(), (Додаток 1 та Додаок 2)

Контрольні питання

1. Що таке консольна Windows-програма?
2. Які стандартні пристрої введення-виведення використовує консоль?
3. Що таке вхідний буфер та екранний буфер консолі?
4. Що таке високорівневий та низькорівнеаий доступ до буферів консолі?
5. Які високорівневі консольні функції Win32 API ви знаєте?
6. Які можливості надають низькорівневі консольні функції?
7. Як визначити стан кнопок миші?
8. Як визначити стан керуючих клавіш клавіатури?
9. Як визначити код символа, введеного з клавіатури?

Лабораторна робота 3.

Тема: Створення віконних програм у Windows

Мета: Ознайомитися із структурою та етапами створення віконних програм для Windows

Короткі теоретичні відомості

Кожна Windows-програма має два суттєвих компоненти: функцію WinMain(), яка ініціалізує вікно, і функцію WindowProc(), яка обслуговує повідомлення Windows. Ця структура становить основу всіх програм Windows.

Функція WinMain() викликається Windows при запуску кожної програми і здійснює необхідну ініціалізацію і налаштування вікна програми. Вона також містить цикл повідомлень для одержання повідомлень із черги повідомлень програми.

Функція WindowProc(), яка інколи має ім'я WndProc() або інше, викликається операційною системою кожного разу, коли повідомлення має передаватися вікну вашої програми. Зазвичай кожне вікно програми має свою функцію WindowProc().

У кожній віконній програмі можна виділити чотири суттєвих компоненти:

1. Реєстрація класу вікна.
2. Створення вікна.
3. Цикл повідомлень.
4. Процедура вікна (віконна функція).

Перших три компоненти складають головну функцію WinMain() а останній є віконною процедурою WindowProc().

Розглянемо усі компоненти віконної програми.

Функція WinMain

Функція WinMain це точка входу до програми Windows. Вона ініціалізує програму, відображує вікно програми на екрані і запроваджує основний цикл повідомлень.

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow);

hInstance дескриптор примірника. Це 32-розрядне число, що ідентифікує примірник нашої програми в середовищі ОС. Цей номер надає Windows, коли програма запускається на виконання.

hPrevInstance дескриптор попереднього примірника, завжди NULL. Це спадок від 16-розрядної Windows.

lpCmdLine параметри командного рядка (програми Windows все ще можуть бути запущені з командного рядка).

NCmdShow визначає, як вікно відображуватиметься. (мінімізоване, максимізоване або приховане).

Функція WinMain завершується, коли отримує повідомлення *WM_QUIT.*

Реєстрація класу вікна

Перед створенням вікна, слід зареєструвати у Windows його клас. Для цього заповнюють структуру WNDCLASSEX і викликають функцію реєстрації класу вікна RegisterClassEx().

typedef struct _WNDCLASSEX {	
UINT cbSize;	// розмір структури
UINT style;	// стиль класу
WNDPROC lpfnWndProc;	// вказівник на віконну функцію
int cbClsExtra;	// додаткові байти класу
int cbWndExtra;	// додаткові байти примірника вікна
HANDLE hInstance;	// дескриптор примірника
HICON hIcon;	// дескриптор іконки
HCURSOR hCursor;	// дескриптор курсору
HBRUSH hbrBackground;	// дескриптор пензля фону
LPCTSTR lpszMenuName;	// вказівник на ім'я меню

LPCTSTR lpszClassName;	// вказівник на ім'я класу
HICON hIconSm;	// мала іконка вікна
} WNDCLASSEX;	

Створюємо змінну WNDCLASSEX wc і заповнюємо структуру WNDCLASSEX:

```
wc.cbSize                 = sizeof(WNDCLASSEX);
wc.style                  = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc            = WindowProc;
wc.cbClsExtra             = 0;
wc.cbWndExtra             = 0;
wc.hInstance              = hInstance;
wc.hIcon                  = LoadIcon(0, IDI_APPLICATION);
wc.hCursor                = LoadCursor(0, IDC_ARROW);
wc.hbrBackground  =GetStockObject(GRAY_BRUSH); wc.lpszMenuName
= 0;
wc.lpszClassName          = "MyWindClass";
wc.hIconSm                = 0;
```

Примітка. Для вибору кольору фону вікна використовують кілька способів.
wc.hbrBackground = (HBRUSH) (COLOR_WINDOW+1); wc.hbrBackground = (HBRUSH) GetStockObject(GRAY_BRUSH); wc.hbrBackground = CreateSolidBrush(RGB(255,255,220));

Є також макрос **GetStockBrush (i),** описаний в заголовковому файлі windowsx.h.

Функція ***реєстрації класу*** вікна:

```
if(!RegisterClassEx(&wc))
{
    MessageBox(NULL, "Window Registration Failed!", "Error!",
MB_ICONEXCLAMATION | MB_OK);
    return 0;
};
```

Створення вікна

Для створення вікна використовують функцію **CreateWindow(),** яка повертає дескриптор створеного вікна. Функція використовує дані із структури **WNDCLASSEX** через вказівник на ім'я зареєстрованого класу lpClassName.

HWND CreateWindow(	
LPCTSTR lpClassName,	// вказівник на ім'я зареєстрованого класу
LPCTSTR lpWindowName,	// вказівник на ім'я вікна
DWORD dwStyle,	// стиль вікна
int x,	// горизонтальна та
int y,	// вертикальна позиція вікна
int nWidth,	// ширина вікна
int nHeight,	// висота вікна
HWND hWndParent,	// дескриптор батьківського вікна (або власника)
HMENU hMenu,	// дескриптор меню або ідентифікатор дочірнього вікна
HANDLE hInstance,	// дескриптор примірника програми
LPVOID lpParam	// вказівник на дату створення вікна
);	

Наприклад:

hWnd = CreateWindow(szClassName, "My First Window", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance, NULL);

Для відображення вікна на екрані комп'ютера слід викликати функцію ShowWindow:

BOOL ShowWindow(hWnd, nCmdShow);

Перший параметр функції ідентифікує вікно і є дескриптором, поверненим функцією CreateWindow(), а

другий – значення nCmdShow, яке передавалося WinMain() і вказує, як вікно з'являється на екрані. Приклади nCmdShow: SW_HIDE, SW_RESTORE, SW_SHOWNORMAL. Можна просто передати значення, одержане від WinMain.

Після цього слід викликати функцію, яка повинна перемалювати (оновити) клієнтську область вікна:
BOOL UpdateWindow(HWND hWnd);

Цикл повідомлень

В кінці головної функції WinMain() створюється цикл повідомлень, призначення якого – діставати повідомлення із черги повідомлень і направляти їх у віконну процедуру для обробки.

Цей цикл зазвичай має вигляд:

```
while(GetMessage(&msg, NULL, 0, 0))    // дістати повідомлення
{
TranslateMessage(&msg);                //транслювати повідомлення
DispatchMessage(&msg);                 // відправити повідомлення
}
```

Цикл виконується весь час роботи програми до одержання повідомлення WM_QUIT, після чого цикл припиняється і програма завершує свою роботу.

Віконна функція

Весь код, який визначає поведінку програми, включається у віконну функцію (процедуру обробки повідомлень). Це функція WindowProc(), на яку посилається **WinMain()** через структуру **WNDCLASSEX.** Windows звертається до цієї функції кожного разу, коли надходить повідомлення для основного вікна програми.

Функція **WindowProc(),** аналізує потік повідомлень із черги, відбирає повідомлення, на які вона повинна відреагувати, і викликає відповідні функції (ділянки коду), відповідальні за обробку тих чи інших подій (обробники подій).

Заголовок віконної функції:

```
LRESULT CALLBACK WindowProc(
HWND hwnd,                  // дескриптор вікна
UINT uMsg,                  // ідентифікатор повідомлення WPARAM
wParam,                     // перший параметр повідомлення LPARAM
lParam                      // другий параметр повідомлення
);
```

Структура віконної функції:

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg, WPARAM wParam,
LPARAM lParam)
{
        switch(msg)
        {
         case WM_CLOSE:
                DestroyWindow(hwnd);
                break;
         case WM_DESTROY:
                PostQuitMessage(0);
                break;
         default:
                return DefWindowProc(hwnd, msg, wParam, lParam);
          }
           return 0;
}
```

Обробка подій у віконній функції

Для обробки подій у віконній функції WindowProc використовують оператор switch, що являє собою список блоків case <мітка>, кожна із яких містить оператори для обробки окремих повідомлень.

Макрос HANDLE_MSG

Замість конструкції **case <мітка>** часто використовують макрос HANDLE_MSG, описаний у заголовковому файлі **windowsx.h:** HANDLE_MSG (hwnd, message, fn).

HANDLE_MSG(hwnd, WM_PAINT, OnPaint);
//case WM_PAINT: OnPaint ();

Цей макрос дає можливість спростити функцію WindowProc і винести усі обробники подій в окремі функції. Оператор **switch** набуває вигляду:

switch(msg){	
HANDLE_MSG(hwnd, WM_PAINT, OnPaint);	
HANDLE_MSG(hwnd, WM_DESTROY, OnDestroy);	
default:	
return(DefWindowProc(hwnd,msg,wParam,lParam));	
}	
Крім цього додається кілька функцій, які є обробниками подій. Наприклад:	
void OnPaint(HWND hwnd){	
char szText[]="Рядок для виводу";	
PAINTSTRUCT ps;	//структура для малювання
HDC hdc = BeginPaint(hwnd,&ps);	//одержання контексту пристрою
TextOut(hdc,5,30,szText,strlen(szText));	//виведення рядка тексту
EndPaint(hwnd,&ps);	//звільнення контексту пристрою
}	

У цьому обробнику використовується функція TextOut, призначена для виведення тексту в робочій області вікна.

TextOut (hdc, x, y, psText, iLength) ;

Тут hdc – дескриптор контексту, який одержується функцією BeginPaint і звільняється функцією EndPaint.

Повідомлення від миші

Повідомлення від миші (наприклад, WM_MOUSEMOVE, WM_LBUTTONDOWN та ін.) мають параметри wParam і lParam, які містять таку інформацію:

```
fwKeys = wParam;          // прапорці клавіш
xPos = LOWORD(lParam);    // горизонтальна позиція курсору
yPos = HIWORD(lParam);    // вертикальна позиція курсору
```

Для розшифровки координат миші використовують змінну типу POINT:

POINT pt;

Обробник повідомлення миші може мати вигляд:

```
case WM_LBUTTONDOWN:
{
   pt.x = LOWORD(lParam); pt.y = HIWORD(lParam);
   wsprintf(str, "Co-ordinates are: X=%i and Y=%i", pt.x, pt.y);
InvalidateRect(hWnd, NULL, TRUE);
}
```

Цей обробник розшифровує координати миші і заносить їх у рядковий масив str.

Функція InvalidateRect оголошує робочу область вікна (або її частину) недійсною

(неправильною), що призводить до її перемальовування.

Завдання для виконання роботи

1. Створити віконну програму для Windows.
2. Замінити іконку програми на одну із наступних: 
3. Замінити курсор вікна на  або інший.

4. Змінити колір фону робочої області на або інший (кольори фону COLOR_APPWORKSPACE, COLOR_BTNSHADOW, COLOR_BTNHIGHLIGHT та ін.).

5. Змінити спосіб відображення вікна на екрані (nCmdShow = SW_SHOWNORMAL, SW_MINIMIZE, SW_MAXIMIZE, SW_SHOWNA, SW_HIDE та ін.).

6. Доповнити віконну функцію програми новими обробниками повідомлень.

a. обробити натискання лівої кнопки миші і вивести в робочій області повідомлення про координати миші.

b. обробити натискання правої кнопки миші і вивести діалогове вікно із текстовим повідомленням про цю подію.

c. оформити обробники повідомлень за допомогою макросу HANDLE_MSG.

Завдання для самостійної роботи

1. Ознайомитися із системними повідомленнями, які надходять від миші та клавіатури.

Контрольні питання

1. Яка структура Windows-програми із графічним інтерфейсом?
2. Яку назву має головна функція віконної програми і які функції вона виконує?
3. Що означає термін «реєстрація класу вікна»?
4. Які дії необхідно виконати у програмі для того, щоб вікно відобразилося на екрані монітора?
5. Що таке цикл повідомлень і як він організується у програмі?
6. Яким чином повідомлення про події в системі потрапляють у програму і як вони обробляються?

9. Яким чином можна налаштувати параметри вікна програми (задати розміри, вибрати іконку, встановити курсор)?

Лабораторна робота 4.

Тема: Використання ресурсів у програмах для Windows

Мета: Ознайомитися із засобами створення та використання ресурсів у програмах для Windows

Короткі теоретичні відомості

Ресурси це заздалегідь визначені дані у програмах Windows, які знаходяться у вико- нуваному файлі у двійковому вигляді і можуть використовуватися при роботі програми. Ці дані не завантажуються в пам'ять при запуску програми. Windows забезпечує функції, що явно завантажують ресурси програми в пам'ять таким чином, що вони можуть викори- стовуватися програмою. Прикладами таких функцій є LoadIcon і LoadCursor, які використовуються при ініціалізації структури класу вікна програми.

До ресурсів програм Windows відносять такі типи ресурсів:

• значки	Icons
• курсори	Cursors
• бітові матриці	Bitmaps
• меню	Menus
• прискорювачі клавіатури	Keyboard accelerators
• діалогові вікна	Dialog boxes
• рядки символів	Character strings
• ресурси користувача	Custom resources

Кожний ресурс має свій опис – сценарій (script), який зберігається у файлі ресурсів із розширенням .rc. Наприклад:

MY_ICON ICON DISCARDABLE "my_icon.ico"
MY_CURSOR CURSOR DISCARDABLE "my_arrow.cur"

Файли ресурсів створюються за допомогою спеціальних редакторів ресурсів, або те- кстових редакторів.

Крім файлу ресурсів до проекту зазвичай включається окремий заголовковий файл **resource.h.** Цей файл містить визначення символічних констант ресурсів, що використо-вуються як у головній програмі, так і у файлі ресурсів. Наприклад:

#define MY_ICON 101
#define MY_CURSOR 104

Сучасні інтегровані середовища розробки програм автоматично формують файли ресурсів і заголовковий файл.

Компіляція файлу ресурсів здійснюється компілятором ресурсів, в результаті чого він перетворюється на об'єктний модуль із розширенням **.res.**

Піктограми

Більшість програм Microsoft Windows включають персональний значок (іконку або icon). Цей значок Windows відображує у лівому верхньому кутку заголовка вікна програ- ми, в меню Пуск, на панелі задач внизу екрану, у папці або на ярлику робочого столу. У віконній програмі Windows цей значок використовується при реєстрації класу вікна:

wc.hIcon = LoadIcon (NULL, IDI_APPLICATION)

Є два варіанти:

1) програма використовує один із стандартних значків Windows IDI_APPLICATION значок програми
   IDI_ASTERISK інформаційний значок
   IDI_EXCLAMATION значок попередження
   IDI_HAND значок помилки
   IDI_QUESTION значок запитання

2) програма використовує свій власний значок

Власний значок будується за допомогою редактора ресурсів і зберігається у файлі із розширенням .ico. Наприклад у Microsoft Visual Studio виконується команда **Insert → Re- source → Icon.** Найкраще мати два значки, розміром 32x32 та 16x16, кожний із яких створюється окремо.

Значок має ідентифікатор, наприклад, IDI_ICON1, MY_ICON та ін.

Для включення значка у вікно програми використовують функції LoadIcon() та LoadImage(). Функція LoadIcon() завантажує значок із файлу програми, а функція LoadI- mage() – із окремого файлу з розширенням **.ICO.**

HICON LoadIcon(

HINSTANCE hInstance, // дескриптор примірника програми

LPCTSTR lpIconName // рядок з іменем значка або ідентифікатор ресурсу

);

При використанні стандартних значків Windows параметр hInstance задають NULL. Другий параметр *lpIconName* вказує на рядок, який містить ім'я ресурсу-значка. Приклади використання функцій:

```
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (101)) ; wc.hIcon = LoadIcon (hInstance, "MyIcon") ;

wc.hIcon = LoadImage(NULL, "alt_icon.ico", IMAGE_ICON, 32, 32,
LR_LOADFROMFILE);
wc.hIconSm = LoadImage(NULL, " alt_icon.ico", IMAGE_ICON, 16, 16,
LR_LOADFROMFILE);
```

Курсори

Програма може використовувати власні курсори, хоча курсори, які поставляє Windows, є цілком адекватними для більшості задач. Використання замовних курсорів миші у програмі подібне до використання власних значків. Замовні курсори загалом мо- нохромні розміром 32□32 пікселів. Курсори створюються за допомогою редактора ресур- сів: Insert→Resource → Cursor. Після створення курсору слід визначити його "гарячу точку" (HotSpot) – точку, яка визначає точну позицію курсору.

Необхідний курсор можна встановити при реєстрації класу вікна за допомогою фун- кції LoadCursor():

HCURSOR LoadCursor(
HINSTANCE hInstance, // дескриптор примірника програми
LPCTSTR lpCursorName // рядок імені або ідентифікатор ресурсу курсору
);

Під час роботи програми курсори можна змінювати за допомогою функції SetCursor().

HCURSOR SetCursor(
HCURSOR hCursor // дескриптор курсору
);

Для використання цієї функції потрібно мати дескриптори курсорів, які можна оде- ржати за допомогою функцій **CreateCursor()** або **LoadCursor().**

Меню

Більшість програм Windows мають "головне меню" (main menu) або "Меню верхньо- го рівня". Елементи головного меню зазвичай викликають "випадаючі" (спливаючі або pop-up) меню.

Меню складаються із пунктів. Є два типи пунктів: пункти-команди і пункти- підменю (Submenu). Можна також визначати численні вкладення підменю: тобто, пункт спливаючого меню може викликати інше спливаюче меню.

Якщо вибраний пункт-команда, то Windows посилає повідомлення WM_COMMAND, а якщо вибраний пункт-підменю, то виводиться наступне спливаюче меню.

Пункти меню можуть бути дозволені (enabled), заборонені (disabled) або недоступні (grayed). Пункти меню можуть бути позначеними значком ∨ (checked). В цьому випадку вони використовуються як перемикачі (check box).

Щоб створити опис чи шаблон меню у Developer Studio, вибирають пункт меню **In- sert → Resource →Menu**. До макета меню, який з'являється додають пункти і визначають їх властивості. У файл ресурсів при цьому заносяться рядки типу:

IDR_MAINMENU MENU DISCARDABLE	
BEGIN	
POPUP "&File"	
BEGIN	
MENUITEM "&Open",	
MENUITEM "&Save",	IDM_FILE_OPEN
MENUITEM SEPARATOR	IDM_FILE_SAVE
MENUITEM "E&xit",	
END	IDM_APP_EXIT
POPUP "&Help"	
BEGIN	
MENUITEM "&Help...",	

MENUITEM "&About MenuDemo...", IDM_APP_HELP END END	IDM_APP_ABOUT

Є кілька способів підключити меню до своєї програми.

1. Підключення меню до програми при реєстрації класу вікна:

wc.lpszMenuName = szMenuName;

2. Завантаження меню функцією LoadMenu. Функція LoadMenu повертає дескриптор ме- ню. Наприклад:

hMenu = LoadMenu (hInstance, TEXT ("MyMenu")) ;

hMenu = LoadMenu (hInstance, MAKEINTRESOURCE (ID_MENU)) ;

Одержаний дескриптор використовується функцією CreateWindow, як її дев'ятий пара- метр:

hwnd = CreateWindow (TEXT ("MyClass"), TEXT ("Window Caption"), WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, hMenu, hInstance, NULL) ;

У цьому випадку меню, визначене дескриптором, перевантажує меню, задане класом вікна.

При виборі пунктів меню Windows посилає віконній функції повідомлення

WM_COMMAND. Це повідомлення вказує, що користувач вибрав дозволений пункт із

меню вікна програми. Молодше слово параметра wParam передає ідентифікатор вибрано- го пункту меню.

Аналізуючи LOWORD (wParam) у WndProc(), можна визначити ідентифікатор пунк- ту меню (ID):

```
case WM_COMMAND: switch(LOWORD(wParam))
{
    case IDM_FILE_OPEN:        // код для File Open
    case IDM_FILE_SAVE:        // код для File Save

case ID_FILE_EXIT: PostMessage(hwnd, WM_CLOSE, 0, 0);
break;

    case IDM_APP_HELP:         // код для Help

case IDM_APP_ABOUT: MessageBox (...);
break;
}
break;
```

Меню можна змінювати під час роботи програми за допомогою функцій:

AppendMenu додає пункт в кінець меню

InsertMenu, InsertMenuItemвставляє новий пункт меню

видаляє пункт меню (також знищує підменю)

RemoveMenu видаляє пункт меню

Є ще цілий ряд функцій для зміни пунктів меню: SetMenuItemInfo, CheckMenuItem, CheckMenuRadioItem, EnableMenuItem, ModifyMenu.

Діалогові вікна

Діалогові вікна – це спеціальні вікна, призначені для діалогу з користувачем. Деякі операції по створенню вікна система робить автоматично (створення та ініціалізація елементів керування, керування передачею фокусу).

Діалогові вікна бувають або "модальні", або "немодальні." Коли програма виводить модальне діалогове вікно, користувач зобов'язаний прийняти певне рішення і закрити діа- логове вікно передбаченим чином (зазвичай натискаючи одну із кнопок OK, Cancel, Retry та ін.). Немодальні вікна можна покинути, не закриваючи їх.

Перший крок по створенню діалогового вікна – це створення сценарію діалогу. Де- талі залежать від компілятора або IDE. У Developer Studio, для прикладу,

вибирають пункт меню Вставка Ресурс →Діалог (Insert → Resource → Dialog) і створюють діалог візу- ально, додаючи до заготовки необхідні компоненти. Файл ресурсу створюється автомати- чно. Для прикладу, файл ресурсу може містити такий текст сценарію діалогу:

```
IDD_ABOUT DIALOG DISCARDABLE 0, 0, 239, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "My About Box"
FONT 8, "MS Sans Serif"
BEGIN
        DEFPUSHBUTTON "&OK",IDOK,174,18,50,14          PUSHBUTTON
"&Cancel",IDCANCEL,174,35,50,14
GROUPBOX "About this program...",IDC_STATIC,7,7,225,52 CTEXT "An example
program showing how to use Dialog
Boxes\r\n\r\nby theForger",IDC_STATIC,16,18,144,33
 END
```

Коли програмі потрібно а при виборі відповідного пункту меню одержує повідом- лення WM_COMMAND із молодшим словом wParam IDM_APP_ABOUT, вона повинна викликати функцію DialogBox для відображення діалогового вікна:

```
int DialogBox(

// дескриптор примірника програми LPCTSTR lpTemplate,
// ідентифікатор шаблону діалогового вікна HWND hWndParent,
// дескриптор батьківського вікна DLGPROC lpDialogFunc
// вказівник процедури діалогового вікна
);
```

Функція повертає числове значення, яке формується після натискання однієї із кно- пок діалогового вікна. Найчастіше це ідентифікатор натиснутої кнопки. Наприклад:

```
case IDM_APP_ABOUT:
{
int ret = DialogBox(GetModuleHandle(NULL),
MAKEINTRESOURCE(IDD_ABOUT), hwnd, AboutDlgProc);
```

```
        if(ret == IDOK){
MessageBox(hwnd, "Dialog exited with IDOK.", "Notice", MB_OK |
MB_ICONINFORMATION);
}
        else if(ret == IDCANCEL){
MessageBox(hwnd, "Dialog exited with IDCANCEL.", "Notice", MB_OK |
MB_ICONINFORMATION);
}
        else if(ret == -1){
MessageBox(hwnd, "Dialog failed!", "Error", MB_OK | MB_ICONINFORMATION);
    }
}
break;
```

Діалогове вікно повинно мати свою віконну процедуру. Вона приймає такі ж пара- метри як і звичайна віконна функція, але має певні відмінності. Наприклад:

```
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,
WPARAM wParam, LPARAM lParam)
{
      switch (message)
      {
      case WM_INITDIALOG :
            return TRUE ;

case WM_COMMAND :
switch (LOWORD (wParam))
{
            case IDOK :
case IDCANCEL :
EndDialog (hDlg, 0) ;
           return TRUE ;
     }
     break ;
  }
  return FALSE ;
}
```

Процедура діалогового вікна має тип BOOL: якщо повідомлення оброблене – повер- тається TRUE (nonzero), якщо ні – FALSE (0).

Для ініціалізації параметрів процедура діалогового вікна оброблює повідомлення

WM_INITDIALOG.

Основним повідомленням, яке потрібно обробити є WM_COMMAND. Це повідом- лення посилає кнопка батьківському вікну при її натисканні мишею або клавішею Space- bar. Ідентифікатор кнопки (наприклад, IDOK) знаходиться у молодшому слові wParam. Для цього повідомлення процедура діалогового вікна викликає функцію EndDialog, яка вказує Windows знищити діалогове вікно.

Завдання для виконання

1. Створити власний значок програми і оформити його як ресурс.
2. Створити власний курсор програми і оформити його як ресурс.
3. Створити власне меню на 2-3 пункти із спливаючими меню на 3-4 пункти. Оформити меню як ресурс. Відредагувати файл ресурсу в текстовому редакторі, додавши новий пункт меню (на 2-3 підпункти).
4. Написати обробники кількох пунктів меню, які виводять повідомлення про обраний пункт.
5. Створити власне діалогове вікно типу:

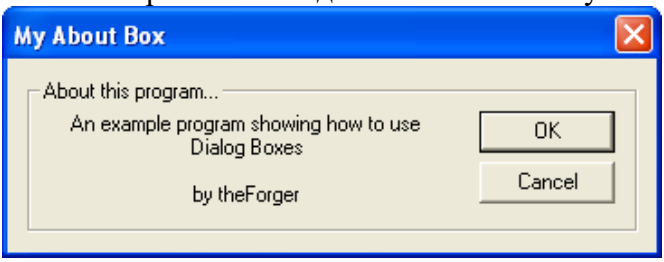

Рис.4.1. Діалогове вікно

У процедурі діалогового вікна обробити події від натискання кнопок OK і Cancel. У віконній процедурі

програми вивести результат натискання кнопок в діалоговому вікні.

6. Додати в діалогове вікно картинку. Для цього спочатку в проект додають ресурс типу Bitmap. Потім в діалогове вікно додають елемент Picture із панелі інструментів. В кінці настроюють властивості (Properties) елемента Picture. Задають ідентифікатор, тип картинки Bitmap і вибирають ідентифікатор ресурсу IDB_Bitmap1.

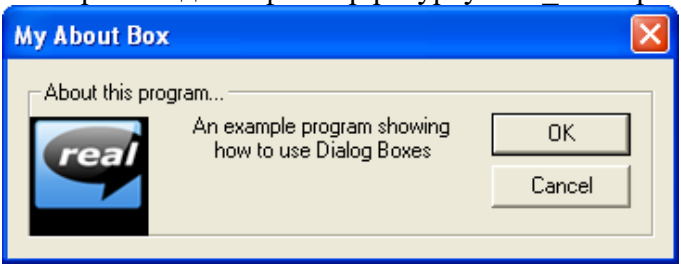

Рис.4.2. Ідентифікатор ресурсу IDB_Bitmap1

7. Використати функцію ShellAbout для виведення стандартного діалогового вікна Windows "О программе".

8. Створити пункт меню, який виводить діалогове вікно MessageBox із кількома кнопка- ми (Yes, No і Cancel). Проаналізувати які значення повертає функція MessageBox і ви- вести повідомлення про те, яка кнопка була натиснута.

Контрольні питання

1. Що таке ресурси і яке значення вони мають для Windows-програм?
2. Яким чином можна створити ресурси і які файли можуть їх містити?
3. Які типи ресурсів ви знаєте і які дані вони містять?

4. Як можна використати у програмах власні іконки або курсори? Як можна змінити кур- сор під час роботи програми?

Лабораторна робота 5.

Тема: Робота з динамічними бібліотеками у програмах для Windows

Мета: Ознайомитися із засобами створення та використання динамічних бібліотек у про- грамах для Windows

Короткі теоретичні відомості

Динамічно зв'язувані бібліотеки (dynamic link library, DLL, динамічні бібліотеки, бібліотечні модулі) – один із найголовніших структурних елементів Microsoft Windows.

Динамічно зв'язані бібліотеки загалом не є безпосередньо виконуваними програма- ми, і вони не отримують повідомлень. DLL – це окремі файли, що містять функції, до яких можуть звертатися програми і інші DLL, щоб виконувати певні завдання. Динамічно зв'я- зана бібліотека починає діяти, лише, коли інший модуль звертається до однієї з функцій бібліотеки.

Динамічні бібліотеки можуть містити набір функцій та/або ресурси. Є два способи виклику функцій DLL із програмного модуля:

Динамічне зв'язування часу завантаження (Неявне завантаження). Динамічні бі- бліотеки завантажуються в пам'ять при завантаженні програми. Програмний модуль здій- снює явні виклики експортованих функцій DLL. Це вимагає, щоб модуль був пов'язаний з бібліотекою імпорту для DLL. Бібліотека імпорту забезпечує операційну систему інфор- мацією, необхідною для завантаження DLL і визначення розташування експортованих функцій при завантаженні програми.

Динамічне зв'язування часу виконання (Явне завантаження). Програмний модуль завантажує необхідні DLL по мірі необхідності під час виконання. Для завантаження біб- ліотеки викликається функція LoadLibrary або LoadLibraryEx. Після того, як DLL завантажена в пам'ять, слід викликати функцію GetProcAddress, щоб одержати адресу експор- тованої функції DLL. Модуль викликає експортовану функцію, використовуючи вказівник на функцію, повернений GetProcAddress. При цьому відпадає потреба в бібліотеці імпорту.

Створення DLL навіть простіше, ніж програми, оскільки вона, як і будь-яка бібліоте- ка, складається лише із набору функцій.

Для створення DLL необхідно:

1. Створити заголовковий файл (наприклад MyLib.h). Необхідні функції відкривають для зовнішнього використання (експортують) за допомогою модифікатора declspec(dllexport). У випадку використання мови C++ забороняють спотворення імен за допомогою extern "C".

```
/*          MyLib.h – заголовковий файл динамічної бібліотеки----------*/
// макрос для експорту функцій без спотворення імен
#ifdef cplusplus
#define EXPORT extern "C" declspec (dllexport) #else
#define EXPORT declspec (dllexport) #endif

// прототипи експортованих функцій
EXPORT void Print(LPCTSTR text);
```

2. Створити файл DLL-модуля (наприклад MyLib.c)

```
/*          MyLib.c – файл динамічної бібліотеки-------------------------*/
#include <windows.h> #include "MyLib.h"
EXPORT void Print(LPCTSTR text)
{
MessageBox(NULL, text, "Title: DLL", MB_OK);
}
```

3. Створити проект: Win-32 Dynamic-Link Library, додати у нього вищенаведені файли і скомпілювати. В

результаті одержують бібліотечні файли MyLib.dll і MyLib.lib.

Використання DLL

Для використання DLL обирають один із способів зв'язування (завантаження).

Неявне завантаження

Створюють проект необхідної програми, додаючи в нього заголовковий файл та виклики бібліотечних функцій, а також повідомляють компонувальнику приєднати файл MyLib.lib (для MS Visual Studio: Project → Settings, закладка Link: в список файлів Ob- ject/Library modules: додати свій файл бібліотеки імпорту). Файл MyLib.lib додають в робочий каталог проекту.

```
/*          MyProg.c – файл програми із використанням MyLib.dll-----*/
#include <windows.h> #include "MyLib.h" int main()
{
Print("Hello!");
Print("The DLL is working!"); Print("By!");
return 0;
}
```

При компіляції програми одержують виконуваний модуль із розширенням .EXE.

Оскільки програмний модуль не містить усіх необхідних для його роботи функцій, то в ньому створюється розділ імпорту, в якому перераховуються імена усіх DLL-модулів та функцій, які використовуються. Ця інформація використовується завантажувальником ОС при запуску програми.

При запуску програми із неявним зв'язуванням завантажувальник ОС створює вірту- альний адресний простір для процесу і проектує в нього виконуваний модуль. Потім ана- лізує розділ імпорту, визначає усі DLL і проектує їх на адресний простір. Якщо DLL ви- кликає інші DLL, то процедура повторюється для кожної із таких DLL.

Необхідні для програми DLL шукаються на диску в такій послідовності:

1. Каталог програми.
2. Поточний каталог процесу.
3. Системний каталог Windows.
4. Основний каталог Windows.
5. Каталоги, визначені змінною оточення PATH.

Якщо необхідна DLL не знайдена, видається відповідне повідомлення.

Явне завантаження

При явному завантаженні бібліотеки імпорту не потрібні, а операції по завантажен- ню необхідної DLL в пам'ять виконуються програмістом. Усі операції проводяться в три етапи:

1. Завантажують необхідні динамічні бібліотеки функцією LoadLibrary(). При успіш- ному завершенні функція повертає значення дескриптора бібліотечного модуля. Якщо функція завершується не успішно, вона повертає NULL.

```
HMODULE hLib;
hLib = LoadLibrary(“MyLib.dll”)
```

2. Визначають вказівники на необхідні функції із DLL. Для цього використовують функцію GetProcAddress. Функція повертає вказівник на бібліотечну функцію. Для при- йому вказівника організовують змінну типу "вказівник на функцію" (див. додаток 1). Оскі- льки функція дає результат типу FARPROC, використовують приведення типів.

```
lpPrint = (MYPROC)(GetProcAddress (hLib,"_Print"));
```

Замість імені функції можна використати її порядковий номер в DLL через макрос

MAKEINTRESOURCE(N), де N – номер функції.

3.Викликають необхідні програмі функції через їх вказівники.

```
lpPrint ("Hello!");
```

4.Звільняють динамічну бібліотеку.

```
BOOL FreeLibrary(hLib);
```

Функція входу/виходу DLL

DLL може мати окрему функцію, яка викликається операційною системою в таких випадках:

• коли процес завантажує DLL;

• коли процес, що використовує DLL, викликає новий потік;

• коли завершається потік, який належить процесу, що використовує DLL;

• коли процес звільняє DLL.

Функція входу/виходу має назву DllMain або DLLEntryPoint:

```
BOOL WINAPI DllMain(
HINSTANCE hinstDLL,          // дескриптор DLL-модуля
DWORD fdwReason,             // причина виклику DLL
LPVOID lpvReserved           // додаткова інформація
);
```

Другий параметр fdwReason може приймати чотири значення:

DLL_PROCESS_ATTACH DLL завантажена в адресний простір процесу

DLL_PROCESS_DETACH DLL відключається від адресного простору процесу

DLL_THREAD_ATTACH створено новий потік

DLL_THREAD_DETACH потік завершується

Функція DllMain містить обробники значень fdwReason, які передаються їй через відповідний параметр.

Завдання для виконання

1.Створити власну DLL, що містить кілька власних функцій.

2.Скомпілювати проект і одержати файли бібліотеки імпорту і динамічної бібліотеки.

3.Створити програму, яка викликає бібліотечні функції із DLL, використовуючи неявне зв'язування (при завантаженні).

4.Створити програму, яка викликає бібліотечні функції із DLL, використовуючи явне зв'язування (під час виконання).

5.Створити DLL ConsoleIO.dll, яка призначена для роботи з консоллю і містить функції PrintStrings, PrintMsg, ConsolePrompt. Текст функцій міститься у файлі Printing.c. На- писати консольну програму для перевірки DLL.(Додаток 3)

Контрольні питання

1.Що таке статична і динамічна бібліотеки? Що в них спільного і що відмінного?

2.Які переваги надає використання динамічних бібліотек?

3.Яким чином динамічна бібліотека зв’язується з основною програмою?

4.Як створити динамічну бібліотеку?

Лабораторна робота 6

Тема: ***Програмування файлових операцій засобами Windows API.***

Мета: Ознайомитися із засобами Win32 API для роботи з файлами. Навчитися використовувати ці функції для програмування файлових операцій.

Короткі теоретичні відомості

Практично в кожній програмі Windows використовують файлові операції.

Найчастіше або початкові дані для програми або дані, одержані в результаті роботи, зберігаються на дисках у вигляді файлів; інколи програмі доводиться створювати ще й тимчасові файли для зберігання поточних даних.

Win32 надає як традиційні засоби роботи з файлами (створення і видалення файлів і каталогів, читання і запис,

пошук, зміна характеристик), так і специфічні засоби, наприклад проектування файлів у пам'ять і сторінкову організацію пам'яті із використанням файлів підкачки).

Створення і відкриття файлу

Функція **CreateFile()** створює новий файл або відкриває існуючий. Ця функція використовується також із такими об'єктами як канали, поштові скриньки, диски, консолі, директорії. Функція повертає дескриптор відкритого файлу.

HANDLE CreateFile(	
LPCTSTR lpFileName,	// вказівник на ім'я файлу
DWORD dwDesiredAccess,	// тип доступу
DWORD dwShareMode,	// тип спільного використання
LPSECURITY_ATTRIBUTES lpSecurityAttributes,	// вказівник на атрибути безпеки DWORD
dwCreationDisposition,	// спосіб створення
DWORD dwFlagsAndAttributes,	// атрибути нового файлу
HANDLE hTemplateFile	// дескриптор шаблону для копіювання
);	// атрибутів

Параметри:

dwDesiredAccess GENERIC_READ | GENERIC_WRITE **dwShareMode** **0** | FILE_SHARE_READ | FILE_SHARE_WRITE **lpSecurityAttributes** NULL або структура SECURITY_ATTRIBUTES

dwCreationDisposition

CREATE_NEW створює новий файл, якщо існує – помилка.
CREATE_ALWAYS перезаписує існуючий файл.
OPEN_EXISTING якщо не існує – помилка.
OPEN_ALWAYS відкриває файл, створює, якщо не існує.
TRUNCATE_EXISTING встановлює нульовий розмір файлу.

dwFlagsAndAttributes

FILE_ATTRIBUTE_NORMAL
FILE_ATTRIBUTE_READONLY тільки для читання.
FILE_ATTRIBUTE_HIDDEN прихований файл FILE_ATTRIBUTE_SYSTEM системний файл
FILE_FLAG_RANDOM_ACCESS відкрити для довільного доступу
FILE_FLAG_DELETE_ON_CLOSE знищити після закриття (для тимчасових файлів).

hTemplateFile дескриптор файлу із правами GENERIC_READ, із якого копіюються атрибути

Закриття файлу:

BOOL CloseHandle(HANDLE hObject); закриває будь-який існуючий дескриптор

Читання файлу

Функція читає задану кількість байтів із файлу, і повертає TRUE, якщо читання успішне.

```
BOOL ReadFile(
HANDLE hFile, // дескриптор файлу
LPVOID lpBuffer, // адреса буферу для прийому
даних DWORD nNumberOfBytesToRead, // кількість байтів для читання
LPDWORD lpNumberOfBytesRead, // адреса числа прочитаних байтів
LPOVERLAPPED lpOverlapped // адреса структури перекриття для асинхронних операцій
// або NULL
);
```

Запис у файл

Функція записує задану кількість байтів у файл, і повертає TRUE, якщо запис успішний.

```
BOOL WriteFile(
HANDLE hFile, // дескриптор файлу
LPCVOID lpBuffer, // вказівник на буфер із даними для запису
DWORD nNumberOfBytesToWrite, // кількість байтів для запису
LPDWORD lpNumberOfBytesWritten, // вказівник на число записаних байтів
LPOVERLAPPED lpOverlapped // вказівник на структуру перекриття для асинхронних
// операцій або NULL
);
```

Видалення файлу

BOOL DeleteFile (LPCTSTR lpFileName);

Копіювання файлу

Windows API має допоміжну функцію CopyFile(), яка копіює файл із заданим іменем у файл із новим іменем:

```
BOOL CopyFile(
LPCTSTR lpExistingFileName, // вказівник на ім'я існуючого файлу
LPCTSTR lpNewFileName, // вказівник на ім'я нового файлу
BOOL bFailIfExists // прапорець для операції, якщо файл існує
);
```

Переміщення (перейменування) файлу

Функція призначена для переміщення або перейменування файлу або каталогу.

Новий файл може бути на іншому диску і навіть в іншій файловій системі. Новий каталог повинен бути на тому ж диску.

```
BOOL MoveFile(
LPCTSTR lpExistingFileName,     // адреса імені існуючого файлу
LPCTSTR lpNewFileName           // адреса нового імені файлу
);
```

Файл із новим іменем не повинен існувати. Якщо файл вже існує, можна користуватись іншою функцією, яка перейменовує існуючі файли:

```
BOOL MoveFileEx(
LPCTSTR lpExistingFileName,     // адреса імені існуючого файлу
LPCTSTR lpNewFileName,          // адреса нового імені файлу
DWORD dwFlags                   // набір прапорців, які визначають як переміщувати файл
);
```

Прапорці це будь-яка комбінація наступних символічних констант: dwFlags

MOVEFILE_REPLACE_EXISTING – замінює існуючий файл

MOVEFILE_COPY_ALLOWED – послідовне виконання CopyFile() і DeleteFile()

MOVEFILE_DELAY_UNTIL_REBOOT – переміщення файлу після перезавантаження комп'ютера

MOVEFILE_WRITE_THROUGH – функція чекає завершення переміщення файлу

Керування розміром файлу

Для ***визначення розміру відкритого файлу*** існує функція:

```
DWORD GetFileSize(
HANDLE hFile, // дескриптор файлу
LPDWORD lpFileSizeHigh          // адреса старшого слова розміру файлу
);
```

Функція повертає молодше подвійне слово розміру файлу. Якщо lpFileSizeHigh = NULL, старше подвійне слово не визначається, інакше lpFileSizeHigh вказує на старше слово розміру файлу.

Є також функція для ***визначення розміру файлу на диску*** за його іменем (невідкритого).

```
DWORD GetCompressedFileSize(
LPCTSTR lpFileName,    // вказівник на ім'я файлу
LPDWORD lpFileSizeHigh        // вказівник на DWORD старшої частини розміру файлу
);
```

Функція повертає розмір стисненого файлу на диску, який підтримує стиснення.

Якщо файл не стиснений, або диск не підтримує стиснення, повертає актуальний розмір файлу.

Для ***зміни розміру відкритого файлу*** служить функція:

```
BOOL SetEndOfFile(
HANDLE hFile  // дескриптор файлу, розмір якого слід змінити
);
```

Функція змінює положення фізичного кінця файлу (EOF). Якщо розмір файлу зменшується, дані в кінці файлу втрачаються. Якщо розмір файлу збільшується, нові дані в кінці файлу не визначені.

Для перевірки наявності файлу або підкаталогу у заданому каталозі призначена функція **FindFirstFile:**

```
HANDLE FindFirstFile(
LPCTSTR lpFileName,    // вказівник на ім'я файлу для пошуку
LPWIN32_FIND_DATA lpFindFileData  // вказівник на повернуту інформацію
);
```

Параметр lpFileName вказує на ім'я файлу (допускаються маски пошуку ? та *).

Параметр lpFindFileData вказує на структуру **WIN32_FIND_DATA**, яка містить повернуту інформацію. Сама функція повертає дескриптор пошуку, який

використовується при наступних пошуках. У випадку помилки пошуку повертається значення **INVALID_HANDLE_VALUE.**

Продовжується пошук файлів за допомогою функції **FindNextFile:**

```
BOOL FindNextFile(
HANDLE hFindFile,        // дескриптор пошуку
LPWIN32_FIND_DATA lpFindFileData   // вказівник на структуру для даних пошуку
);
```

Завершується пошук функцією:

```
BOOL FindClose(
HANDLE hFindFile         // дескриптор пошуку
);
```

Керування каталогами

Для ***створення каталогу*** використовують функцію:

```
BOOL CreateDirectory(
LPCTSTR lpPathName,   // вказівник на рядок шляху каталогу
LPSECURITY_ATTRIBUTES lpSecurityAttributes   // вказівник на дескриптор безпеки
);
```

Дескриптор безпеки це структура SECURITY_ATTRIBUTES, яка задає атрибути безпеки для каталогу, що створюється. Якщо lpSecurityAttributes = NULL, використовуються параметри за умовчанням. Файлова система диску, на якому створюється каталог, повинна підтримувати атрибути безпеки.

Для ***видалення каталогу*** використовують функцію:

```
BOOL RemoveDirectory(
LPCTSTR lpPathName    // адреса імені каталогу, який потрібно видалити
);
```

Видалити можна лише порожній каталог.

Одержати поточний каталог можна за допомогою функції:

```
DWORD GetCurrentDirectory(
DWORD nBufferLength,  // розмір буфера імені каталогу в символах
LPTSTR lpBuffer           // адреса буфера для поточного каталогу
```

```
);
```

Функція повертає довжину рядка з іменем каталогу. У випадку помилки повертає нуль. Якщо буфер недостатній для прийому імені каталогу, повертає необхідний розмір буфера.

Наступна функція ***встановлює поточний каталог*** для процесу:

```
BOOL SetCurrentDirectory(
LPCTSTR lpPathName    // адреса імені нового поточного каталогу
);
```

Шлях може бути абсолютним (C:\masm32\bin) або відносним (\\Thunder\Stuff).

Контрольні питання

1.Як організована робота з файлами стандартними засобами мови С?

2.Які функції Win API для роботи з файлами ви знаєте?

3.Для чого використовують функцію CreateFile() і які вона має параметри?

4.Як діє функція CreateFile() у випадку, якщо ім'я створюваного файлу співпадає з існуючим на диску?

5.Які функції Win API застосовують для читання-запису даних у файли?

Завдання для самостійної роботи

1.Ознайомитись із функціями WinAPI ***GetSystemTime(), GetLocalTime()*** та структурою SYSTEMTIME.

2.Створити програму для визначення часу копіювання файлу із одного каталогу в інший. (Додаток 4 та Додаток 5)

ДОДАТКИ

Додаток 1
Скелет консольної програми

```
HANDLE hIn, hOut;
DWORD Read, Written;
DWORD Numb = 1;
char Name[] = "Using of Console functions\n\n";

FreeConsole();
AllocConsole();
hIn = GetStdHandle(STD_INPUT_HANDLE); hOut =
GetStdHandle(STD_OUTPUT_HANDLE);

WriteConsole(
hOut,   // handle to a console screen buffer
Name,  // pointer to buffer to write from
strlen(Name),  // number of characters to write
&Written,       // pointer to number of characters written
NULL  // reserved

...
);

ReadConsole(
hIn,     // handle of a console input buffer
Buf,     // address of buffer to receive data
Numb, // number of characters to read
&Read,          / address of number of characters read
NULL  // reserved

);

return 0;
}
```

Додаток 2

Приклад використання низькорівневих функцій

```
HANDLE hIn, hOut;
DWORD Read, Written;
DWORD ToRead = 1;
INPUT_RECORD InRec;
COORD Coor;
Coor.X = 0;
Coor.Y = 5;
/* Читаємо запис вхідного буферу, якщо KEY_EVENT – виводимо його поля */
while ((int)InRec.Event.KeyEvent.uChar.AsciiChar != 27)
{
ReadConsoleInput(hIn, &InRec, ToRead, &Read);
SetConsoleCursorPosition(hOut, Coor);
if (InRec.EventType == KEY_EVENT)
{         // стираємо рядок і виводимо поля
FillConsoleOutputCharacter(hOut,' ',80,Coor,&Written);
printf("%u %u %u %u %#x - %c %#x",
InRec.Event.KeyEvent.bKeyDown,
InRec.Event.KeyEvent.wRepeatCount,
InRec.Event.KeyEvent.wVirtualKeyCode,
InRec.Event.KeyEvent.wVirtualScanCode,
InRec.Event.KeyEvent.uChar.UnicodeChar,
InRec.Event.KeyEvent.uChar.UnicodeChar,
InRec.Event.KeyEvent.dwControlKeyState);
};
};
```

Додаток 3

Оголошення вказівника на функцію

Вказівник на функцію це змінна, яка вказує на функцію певного типу із заданим ти- пом аргументів. Отже повинна існувати функція, на яку буде посилатися вказівник. Нехай треба зробити вказівник на функцію: int Calc (float x, char s) Для оголошення вказівника на функцію використовують конструкцію виду:
return_type (*pointer_name)(list_of_parameter_types);

Наприклад:
int (*ptrToFunc) (float, char); ptrToFunc = Calc;

Для оголошення вказівника на функцію вводять тип: вказівник на функцію, опи- сують змінну введеного типу і присвоюють їй значення – адресу функції
typedef int (*MYPTR) (float, char); MYPTR ptrToFunc;
ptrToFunc = Calc;

Оголошення вказівника суміщають із присвоєнням йому значення:
int (*ptrToFunc) (float, char) = Calc;

Примітка. Середовище DEV-C++ використовує компілятор GCC, який зазвичай створює бібліотеку імпорту з розширенням ".a". Цей файл потрібно додати до проекту основної програми.

Додаток 4

Програма копіювання файлів cpC.c (з використанням функцій C)

```
/* Basic cpC file copy program. C library functions. */
/* cp file1 file2: Copy file1 to file2. */

#include <windows.h> #include <stdio.h> #include <errno.h>
#define BUF_SIZE 256

int main (int argc, char *argv [])
{
FILE *in_file, *out_file; char rec [BUF_SIZE];
size_t bytes_in, bytes_out; if (argc != 3) {
printf ("Usage: cpC file1 file2\n"); return 1;
}
in_file = fopen (argv [1], "rb"); if (in_file == NULL) {
perror (argv [1]);
return 2;
}
out_file = fopen (argv [2], "wb"); if (out_file == NULL) {
perror (argv [2]);
return 3;
}
/* Process the input file a record at a time. */

while ((bytes_in = fread (rec, 1, BUF_SIZE, in_file)) > 0) {
bytes_out = fwrite (rec, 1, bytes_in, out_file);
if (bytes_out != bytes_in) { perror ("Fatal write error."); return
4;
}
}
fclose (in_file);

fclose (out_file); return 0;
```

}

Додаток 5

Програма копіювання файлів cpW.c (з використанням функцій WinAPI)

```
/* Basic cpW file copy program. Win32 functions. */
/* cpW file1 file2: Copy file1 to file2. */
#include <windows.h> #include <stdio.h> #define BUF_SIZE 256
int main (int argc, LPTSTR argv [])
{
HANDLE hIn, hOut;
DWORD nIn, nOut;
CHAR Buffer [BUF_SIZE];
if (argc != 3) {
printf ("Usage: cpW file1 file2\n"); return 1;
}
hIn = CreateFile (argv [1], GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
if (hIn == INVALID_HANDLE_VALUE) {
printf ("Cannot open input file. Error: %x\n", GetLastError ());
return 2;
}
hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (hOut == INVALID_HANDLE_VALUE) {
printf ("Cannot open output file. Error: %x\n", GetLastError ());
return 3;
}
while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn >
0) { WriteFile (hOut, Buffer, nIn, &nOut, NULL);
if (nIn != nOut) {
printf ("Fatal write error: %x\n", GetLastError ()); return 4;}}
CloseHandle (hIn); CloseHandle (hOut); return 0;}
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рисований О.М. Системне програмування : підручник. Х. : НТУ “ХПІ”, 2010. 912 с.
2. Рисований О. М. Системне програмування : навч. посіб. Для студентів напрямку «Комп’ютерна інженерія» вищих навчальних закладів у 4 ч. Ч. І. Основи асемблера та його використання. / О. М. Рисований. Харків : «Слово», 2015. 336 с.
3. Рисований О. М. Системне програмування : навч. посіб. Для студентів напрямку «Комп’ютерна інженерія» вищих навчальних закладів у 4 ч. Ч.2. Розширені можливості програмування на асемблері. Харків : «Слово», 2015. 224 с.
4. Паламар А. Комп'ютерна система для моніторингу параметрів джерел безперебійного живлення на основі технології Internet of Things. Матеріали IV Міжнародної науковотехнічної конференції "Теоретичні та прикладні аспекти радіотехніки, приладобудування і комп’ютерних технологій", 20-21 червня 2019 року: збірник тез доповідей. Тернопіль : ФОП Паляниця В. А., 2019. С. 208–209.
5. Паламар М. І., Стрембіцький М. О., Паламар А. М. Проектування комп’ютеризованих вимірювальних систем і комплексів : навчальний посібник. Тернопіль : ТНТУ, 2019. 150 с.
6. Паламар А. М., Пастернак Ю. В., Паламар Я. М. Двохпроцесорна інформаційновимірювальна система керування пристроєм безперебійного електроживлення.
Матеріали III Міжнародної науково-технічної конференції молодих учених та студентів "Актуальні задачі сучасних технологій", 19-20 листопада 2014 р. Тернопіль : ТНТУ, 2014. С. 211–212.