МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ШТУЧНОГО ІНТЕЛЕКТУ ТА РОБОТОТЕХНІКИ
КАФЕДРА ПРОГРАМНИХ І КОМП’ЮТЕРНО-ІНТЕГРОВАНИХ ТЕХНОЛОГІЙ

Методичні вказівки з дисципліни

ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

(Лабораторний практикум. Частина перша)

Для студентів інституту штучного інтелекту та робототехніки

Перший (бакалаврський) рівень вищої освіти

Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології

Освітньо-професійна програма: Комп'ютерні технології автоматизації;

Одеса 2022

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ ШТУЧНОГО ІНТЕЛЕКТУ ТА РОБОТОТЕХНІКИ

КАФЕДРА ПРОГРАМНИХ І КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ ТЕХНОЛОГІЙ

Методичні вказівки з дисципліни

ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

(Лабораторний практикум. Частина перша)

Для студентів інституту штучного інтелекту та робототехніки

Перший (бакалаврський) рівень вищої освіти

Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології

Освітньо-професійна програма: Комп'ютерні технології автоматизації.

Схвалено на засіданні кафедри ПКІТ протокол №7 від 26.01.2022 р.

Одеса 2022

Методичні вказівки з дисципліни «Обчислювальна техніка та комп'ютерні технології» (Лабораторний практикум. Частина перша) для студ. спец. 151 – «Автоматизація та комп'ютерно-інтегровані технології» ден. та заоч. форм навч./уклад.: В.О. Давидов - Одеса: ОП, 2022. - 133 с.

Укладачі: **В.О. Давидів**, Канд. техн. наук

ЗМІСТ

загальні вказівки 5
Корисні посилання 6
Лабораторна робота №1 7
Теоретичні відомості 7
Програмна частина 12
Індивідуальне завдання 18
Контрольні питання 18
Лабораторна робота №2 19
Теоретичні відомості 19
Програмна частина 20
Індивідуальне завдання 32
Контрольні питання 33
Лабораторна робота №3 34
Теоретичні відомості 34
Програмна частина 36
Індивідуальне завдання 42
Контрольні питання 43
Лабораторна робота №4 44
Теоретичні відомості 44
Програмна частина 48
Індивідуальне завдання 60
Контрольні питання 60
Лабораторна робота №5 61
Теоретичні відомості 61
Програмна частина 64
Лабораторна робота №6 78
Теоретичні відомості 78
Програмна частина 84
Лабораторна робота №7 .... **Ошибка! Закладка не определена.**
Теоретичні відомості .... **Ошибка! Закладка не определена.**
Програмна частина .... **Ошибка! Закладка не определена.**

ЗАГАЛЬНІ ВКАЗІВКИ

Цей курс лабораторних робіт орієнтований використання IDE Visual Studio C++ 2019 Community Edition. Це безкоштовна IDE, що вільно розповсюджується для навчальних цілей.

Докладно інформацію про те, як завантажити та встановити IDE викладено у «Методичні вказівки до виконання лабораторних робіт дисципліни Програмування та теорія алгоритмів. Частина 1 2020.

Інформація про те, як створювати візуальні програми на базі MFC викладена у «Методичні вказівки до виконання лабораторних робіт дисципліни Програмування та теорія алгоритмів. Частина 2» 2021.

Слід зазначити, що за останні десятиліття і технологія програмування і мови програмування значно еволюціонували і деякі приклади старих програм тепер навіть не компілюються так як вимагають дещо іншого підходу. Якщо ви зіткнетеся з подібною ситуацією, коли начебто робочий приклад з якогось джерела навіть не компілюється, постарайтеся уважно вивчити код прикладу. Можливо, в нього необхідно внести деякі зміни і все запрацює. Приклади таких типових змін наведено нижче.

Приклад старого коду, який не компілюється у VC 2019:

```
char szBuf[16];
sprintf(szBuf,"%g", m_dInFlowRate);
SetDlgItemText(IDC_IN_VALUE, szBuf);
```

Еквівалент у VC 2019:

```
CString str;
str.Format(_T("%g"), m_dInFlowRate);
SetDlgItemText(IDC_IN_VALUE, str);
```

або

```
CString str;
str.Format(TEXT("%g"), m_dInFlowRate);
SetDlgItemText(IDC_IN_VALUE, str);
```

КОРИСНІ ПОСИЛАННЯ

https://ravesli.com/uroki-cpp/	Подано понад 200 безкоштовних уроків з програмування мовою С++. Онлайн курси програмування з нуля для початківців, де розглядаються основи та тонкощі мови програмування С++. Безкоштовне навчання програмування, а саме підручник з практичними завданнями та тестами. Неважливо, чи маєте Ви досвід чи ні, ці уроки з програмування допоможуть Вам почати створювати, компілювати та налагоджувати програми мовою С++ у різних середовищах розробки Visual Studio, Code::Blocks, Xcode або Eclipse. Безліч прикладів та докладних роз'яснень. Добре підійдуть як для новачків (чайників), так і для більш просунутих.
https://evileg.com/ru/knowledge/	Соціальна мережа програмістів із сервісами статей, форуму, тестувань та можливості розміщення вакансій.

ЛАБОРАТОРНА РОБОТА №1

Мета роботи: відновити навички програмування з прикладу розв'язання задачі чисельного інтегрування

Завдання:
1) Вивчити теоретичні відомості.
2) Створити прототип програми чисельного інтегрування.
3) реалізувати індивідуальне завдання.

Теоретичні відомості

Перед тим, як розпочати опис завдання, нагадаємо раніше вивчені елементи з інших дисциплін, але які будуть задіяні в практичній роботі.

Для початку пояснимо термін «таблична функція». Це функція, значення точок, якою представлені у вигляді таблиці (табл. 1.1).

Таблиця № 1.1 - Таблична функція

x_i	0	1	2	3
y_i	0	1	4	9

За даними таблиці видно, що функція представлена квадратичною залежністю $y = x^2$.

Також ви можете мати справу з терміном «тимчасовий ряд». Це зібраний у різні моменти часу статистичний матеріал про значення будь-яких параметрів (у найпростішому випадку одного) досліджуваного процесу. Кожна одиниця статистичного матеріалу називається виміром чи відліком. У тимчасовому ряді для кожного відліку має бути вказано час виміру або номер виміру по порядку.

Чисельне інтегрування у роботі розглядається з прикладу трьох видів функцій: пряма, парабола і експонента.

Пряма є окремий випадок лінійної функції. У загальному вигляді рівняння прямої лінії:

$y=kx + b$,

де k – коефіцієнт при незалежній змінній x, який дорівнює тангенсу кута нахилу між кривою та віссю абсцис;
b – вільний коефіцієнт, що дозволяє переміщати графік вздовж осі ординат.

Окремий випадок: y=kx

На малюнку 1.1а представлені графіки функцій:

y = 2x - 1 (графік 1)
y = - x +4 (графік 2)

На малюнку 1б представлені графіки:

y = x (графік 1)
y = - x/2 (графік 2)

Квадратична функція також є окремим випадком лінійної функції. У загальному вигляді квадратична функція описується виразом:

y = ax2 + b,

де a – коефіцієнт при незалежній змінній x, який дорівнює тангенсу кута нахилу між кривою та віссю абсцис;
b – вільний коефіцієнт, що дозволяє переміщати графік вздовж осі ординат.

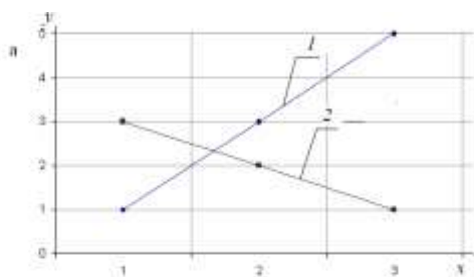
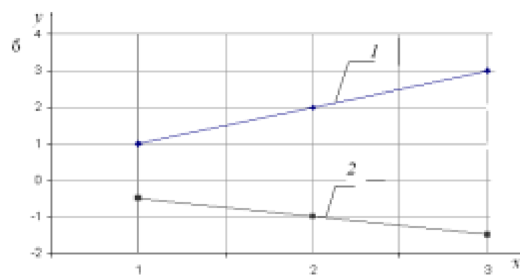



Рисунок 1.1 – Графіки прямих функцій

Окремий випадок: y = ax2

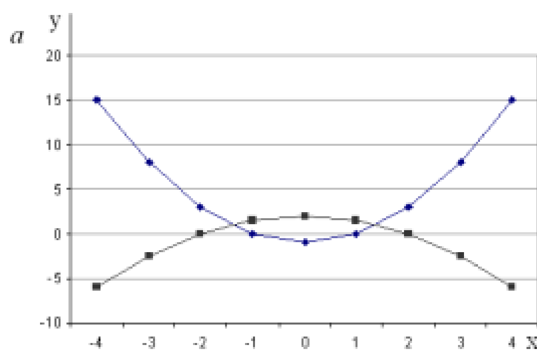
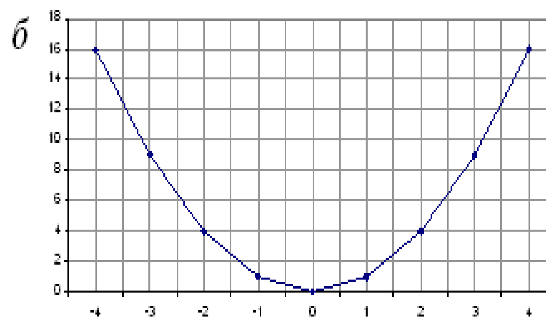



Рисунок 1.2 – Графіки квадратичних функцій

На рис 1.2а представлені графіки:

y = x2 - 1 (графік 1)
y = - x2/2 + 2 (графік 2)

На рис. 1.2б представлений графік кривої y = x2
Загальний вигляд експоненційної функції:

y = ekx +b,

де k – коефіцієнт при незалежній змінній x;
b -Вільний коефіцієнт, який дозволяє переміщувати графік вздовж осі ординат.
Окремий випадок:

y = *ekx*

На рис 1.3 представлені графіки:

y = ex (графік 1)
y = e-0,95x (графік 2)

Перейдемо поняття інтеграла. Що таке невизначений інтеграл? Це первісна функція за аргументом. Якщо є функція $y = f(x)$, то її первісна має вигляд $Y + C = \int f(x) \cdot dx$, де C - Довільна постійна. Дія зворотне інтегрування - диференціювання, тобто $(Y + C)' = f(x)$.

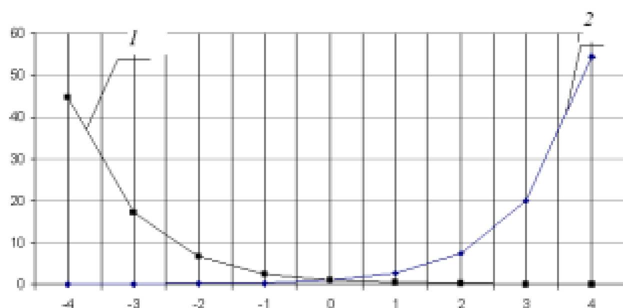


Рисунок 1.3 – Графіки експонентних функцій:

Невизначений інтеграл із заданими межами інтегрування називають певним інтегралом. Тобто щоб перейти від невизначеного інтеграла до певного інтегралу, необхідно задати межі інтегрування. Для аналітичного розрахунку інтеграла використовується формула Ньютона – Лейбниця.

$$Y = \int_a^b f(x) \cdot dx = F(x)|_a^b = F(b) - F(a).$$

Важливо розуміти, що невизначений інтеграл це функція, а певний число.

Як правило, у будь-якому математичному довіднику можна знайти таблицю інтегралів, у якій наводяться результати взяття невизначених інтегралів від типових функцій.

На жаль, на практиці існують функції, від яких не можна взяти інтеграл (наприклад, $y = \frac{\sin(x)}{x}$). Також зустрічаються завдання, коли аналітичний вид функції невідомий, а функція задана в табличному вигляді. У цих випадках інтеграл розраховують чисельним способом (ми розглядатимемо метод правих прямокутників).

Геометричний сенс певного інтеграла безперервної функції $y = f(x)$ на інтервалі $[a,b]$ – є площа, обмежена графіком цієї функції, прямими $x = a, x = b$, і віссю абсцис.

Розглянемо розрахунок певного інтеграла методом прямокутників на прикладі трьох функцій: прямої, квадратичної та експоненти. Для зручності візьмемо окремі випадки $y = x$, $y = x^2$ і $y = e^x$

Розіб'ємо графіки функцій на n – прямокутників (рис. 1.4). Наближена площа цієї кривої дорівнює сумі площ прямокутників, при цьому чим менше $\Delta x = x_i - x_{i-1}$, тим точніше значення площі під кривою $y = f(x)$.

З малюнка 1.4 видно, що площа одного прямокутника дорівнює $y(x_i) \cdot (x_i - x_{i-1})$, площа кривої на відрізку $[a,b]$ дорівнює $\sum_{i=1}^{n} y(x_i) \cdot (x_i - x_{i-1})$, при цьому $x_0 = a, x_n = b$. Отже, інтеграл функції $y = f(x)$ на відрізку $[a,b]$ дорівнює $\sum_{i=1}^{n} y(x_i) \cdot (x_i - x_{i-1})$.

Цей підхід називається методом лівих прямокутників. Зверніть увагу на те, що щодо аналізованої точки x_i прямокутник будується ліворуч. Існують також метод правих прямокутників та центральних. В останньому методі центр прямокутника розташований у точці x_i. Це дозволяє частково компенсувати позитивні та негативні похибки щодо інтеграла.

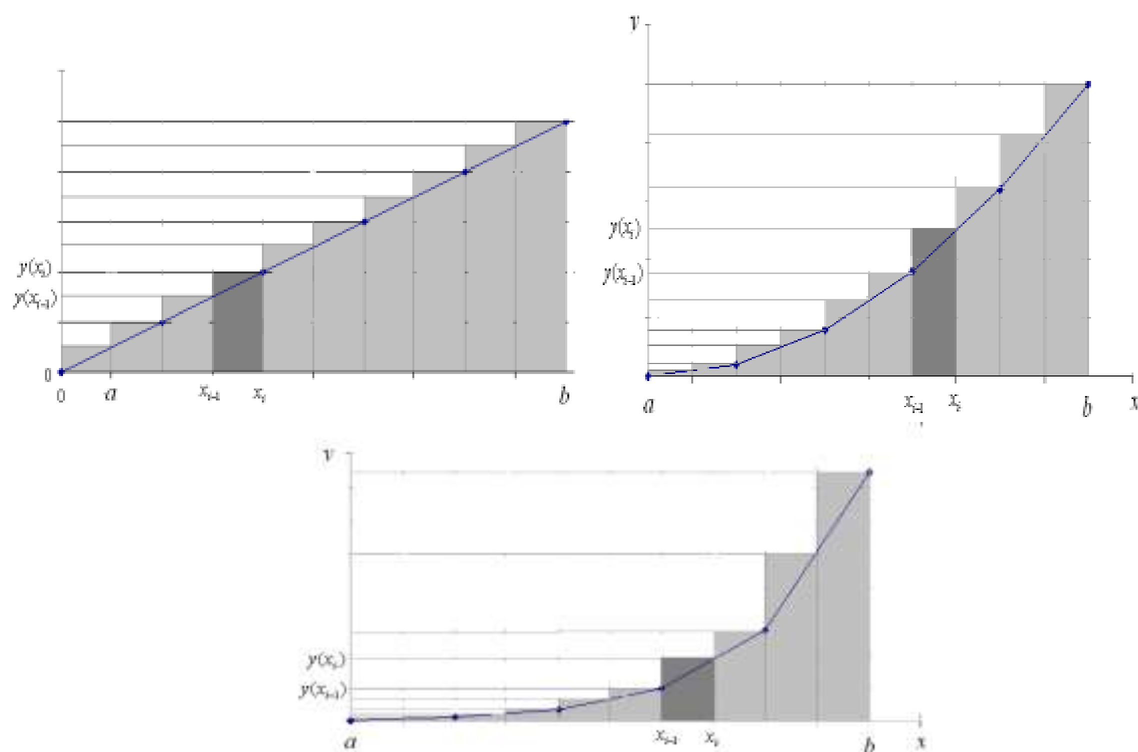


Рисунок 1.4 – Чисельне визначення інтеграла методом лівих прямокутників

У розглянутих прикладах метод лівих прямокутників призведе до того, що розраховане значення інтеграла буде більшим за істинне. Метод правих прямокутників дав зворотний результат. Розраховане значення було менше істинного.

Для подальшої роботи для визначеності приймемо, що інтервал $[a,b]$, На якому визначені функції і на якому здійснюватиметься інтегрування дорівнює [0, 5].

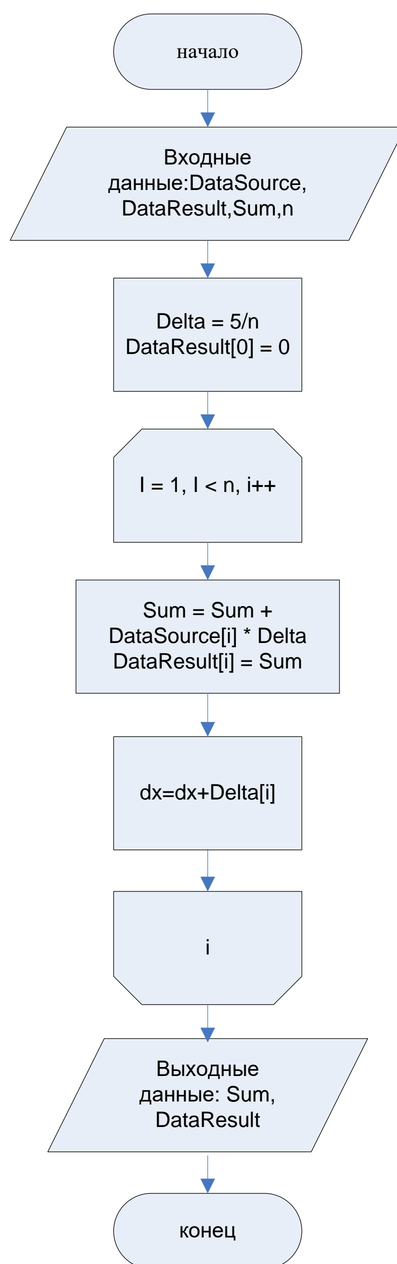


Рисунок 1.5 – Алгоритм розрахунку площі методом прямокутників

На рис. 1.5 наведено алгоритм чисельного обчислення інтеграла.

Програмна частина

Розглянемо крок за кроком створення нашого додатка для чисельного розрахунку інтеграла.

Створення проекту

Запустіть IDE та створіть новий проект типу «Додаток MFC». У вікні задайте ім'я проекту«Integral» і вкажіть місце, де його буде створено в полі Розташування. Наприклад, нехай це може бути папка D:\MyProject. Натисніть кнопкуСтворити.

У вікні, що з'явилося в полі Тип програми вкажіть На основі діалогових вікон і натисніть Готово.

Нагадуємо, що як на заняттях у ВНЗ, так і під час роботи вдома, доцільно створити для роботи окрему папку та надалі вказувати її при створенні всіх наступних проектів. Налаштувати шлях до папки за промовчанням можна в менюКошти → Параметри → Проекти та рішення → Розташування (Рис. 1.6).

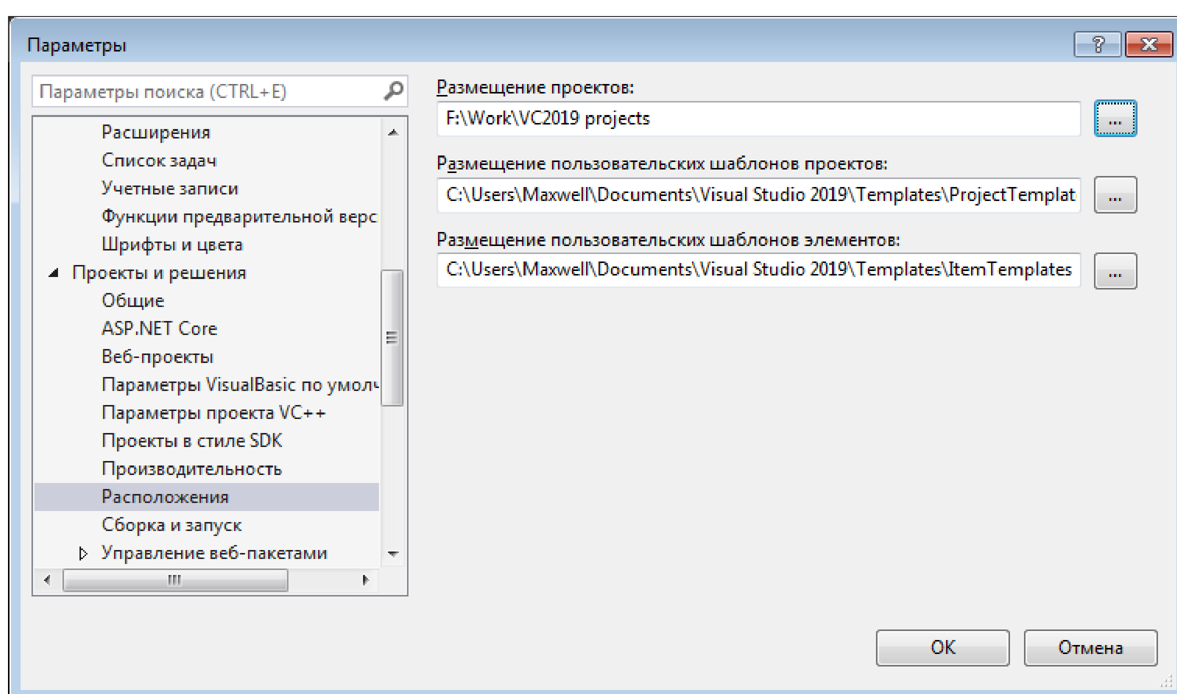


Рисунок 1.6 – Налаштування шляху розміщення проектів за замовчуванням

Також нагадуємо, що копіювання файлів проектів без чіткого розуміння того, що і куди ви копіюєте неприпустимо. Як ви пам'ятаєте з першого курсу, кожен проект складається з багатьох файлів. У різних проектах є базові файли, що мають одну й ту саму назву, наприклад,resource.h. Копіюючи файли з одного проекту до іншого, ви можете переписати такий базовий файл і цим порушити структуру проекту. Після цього він перестане запускатись. Тому варіант«Скопіюю у товариша файли, а інтерфейс залишу свій» як правило, призводить до того, що проект доводиться викидати і розпочинати роботу із самого початку.

Розробка інтерфейсу користувача

Щоб відкрити форму діалогового вікна для редагування, виберіть у браузері рішень у папці Файли ресурсів файл Ім'яПроекта.rc2. У вікні ресурсів, що з'явилося, в розділі Dialog Виберіть відповідну форму.

Якщо у вас на екрані відсутня панель інструментів, увімкнути її можна в менюВид → Панель інструментів.

Модифікуйте вікно інтерфейсу користувача, так як це показано на рис. 1.7. Інтерфейс містить дві області для малювання (елементиPicture). Верхній - надайте ідентифікаторIDC_DRAW_AREA, нижній - IDC_RESULT. У верхній області малюватимуться графіки вихідної функції та її похідної. У нижній – легенда.

Три кнопки (елементи Button) привласніть ідентифікатори: IDC_BTN_EXPFUNC, IDC_BTN_POWFUNC, IDC_BTN_LINEFUNC. Поля для редагування (елементEditBox) привласніть ідентифікатор IDC_INTEGRAL. У нього виводитиметься розраховане значення інтеграла.

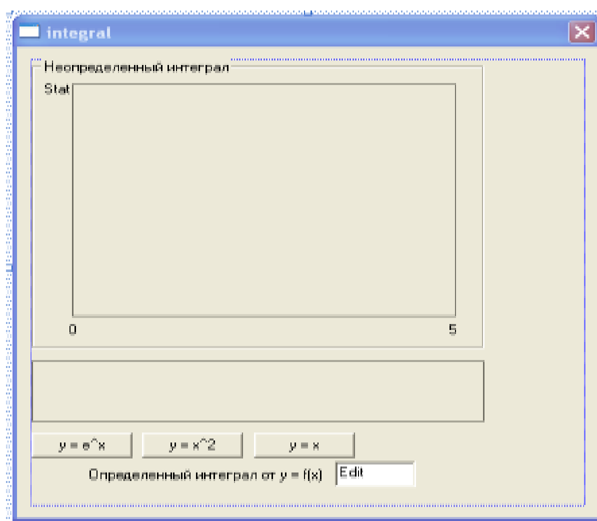


Рисунок 1.7 – Вигляд інтерфейсу користувача.

У верхньому лівому куті інтерфейсу розташований елемент Static text з ідентифікатором IDC_Y. У цей елемент буде виводитися максимальне значення координати для обох графіків округлене до цілих.

Інші деталі інтерфейсу побудовані за допомогою елементівGroup Box і Static text. Їхні ідентифікатори можна залишити без зміни.

Далі необхідно прив'язати змінну, що дозволяє керувати вмістом осередкуEditBox. Виберіть у менюПроект → Майстер класів → Змінні члени. В поліІм'я класу вкажіть CIntegralDlg. У полі Змінні члени виберіть IDC_INTEGRAL і натисніть кнопкуДодати змінну. Заповніть поля діалогу таким чином:

Поле	Ідентифікатор змінної
Ім'я	m_dInt
Категорія	Значення
Тип	double

Далі створіть обробники кнопокOnBtnExpfunc(), OnBtnPowfunc() та OnBtnLinefunc() двічі клацнувши мишкою на відповідних кнопках на інтерфейсі.

Написання програмного коду

Оскільки дана робота перша з двох робіт з відновлення навичок програмування, деякі моменти ми будемо реалізовувати максимально примітивно. Так ми не будемо використовувати динамічну пам'ять, покажчики, функції користувача та ін.

Почнемо з модифікації файлу IntegralDlg.h.

```
#define DATA_SIZE 100
// оголошення глобальної константи, яка
// визначає кількість елементів у табличній функції
```

```
// Діалогове вікно CIntegralDlg
class CIntegralDlg : public CDialogEx
{
// Створення
public:
	CIntegralDlg(CWnd* pParent = nullptr); // Стандартний конструктор

	void Draw();                     //оголошуємо функцію, в якій будемо
	                                 // реалізовувати побудову графіків
	double m_dDataSource[DATA_SIZE]; //масиви, у яких зберігатиметься значення
	double m_dDataResult[DATA_SIZE]; // функції та її інтеграла, відповідно
	double dSum;                     // Оголошуємо змінну для підсумовування
	double Ymin;                     // оголошення мінімального та максимального
	double Ymax;                     // значення масиву відповідно
```

Далі модифікуйте обробники кнопок.

```
void CIntegralDlg::OnBnClickedBtnExpfunc()
{
	// TODO: додайте свій код обробника повідомлень
	UpdateData(true);

	double dX = 0;
	double dDelta = 5. / DATA_SIZE;         // задаємося кроком

	// для коректного малювання графіків функцій необхідно знати їх максимальне
	// і мінімальне значення
	// ініціалізуємо змінні для мінімуму та максимуму
	Ymin = exp (dX);
	Ymax = exp(dX);

	// формуємо вихідну функцію
	for (inti = 0; i < DATA_SIZE; i++)
	{
		m_dDataSource[i] = exp(dX);
		dX += dDelta;
		if (Ymin > m_dDataSource[i]) Ymin = m_dDataSource[i];
		if (Ymax <m_dDataSource[i]) Ymax = m_dDataSource[i];
	}
	// безпосередньо розрахунок інтеграла
	dSum = 0;
	m_dDataResult[0] = 0;
	for (inti = 1; i < DATA_SIZE; i++)
	{
		dSum += m_dDataSource[i] * dDelta;
		m_dDataResult[i] = dSum;
		if (Ymin > m_dDataResult[i]) Ymin = m_dDataResult[i];
		if (Ymax <m_dDataResult[i]) Ymax = m_dDataResult[i];
	}
	m_dInt = dSum;                   //Виведення значення інтеграла на інтерфейс
	UpdateData(false);
	Draw();             // виклик функції малювання графіка
}

void CIntegralDlg::OnBnClickedBtnPowfunc()
{
	// TODO: додайте свій код обробника повідомлень
	UpdateData(true);

	double dX = 0;
```

```
    double dDelta = 5. / DATA_SIZE;
    Ymin = exp(dX);
    Ymax = exp(dX);
    for (inti = 0; i < DATA_SIZE; i++)
    {
        m_dDataSource[i] = dX * dX;
        dX += dDelta;
        if (Ymin > m_dDataSource[i]) Ymin = m_dDataSource[i];
        if (Ymax <m_dDataSource[i]) Ymax = m_dDataSource[i];
    }
    dSum = 0;
    m_dDataResult[0] = 0;
    for (inti = 1; i < DATA_SIZE; i++)
    {
        dSum += m_dDataSource[i] * dDelta;
        m_dDataResult[i] = dSum;
        if (Ymin > m_dDataSource[i]) Ymin = m_dDataSource[i];
        if (Ymax <m_dDataSource[i]) Ymax = m_dDataSource[i];
    }
    m_dInt = dSum;
    UpdateData(false);
    Draw();
}

void CIntegralDlg::OnBnClickedBtnLinefunc()
{
    // TODO: додайте свій код обробника повідомлень
    UpdateData(true);

    double dX = 0;
    double dDelta = 5. / DATA_SIZE;
    Ymin = exp(dX);
    Ymax = exp(dX);
    for (inti = 0; i < DATA_SIZE; i++)
    {
        m_dDataSource[i] = dX;
        dX += dDelta;
        if (Ymin > m_dDataSource[i]) Ymin = m_dDataSource[i];
        if (Ymax <m_dDataSource[i]) Ymax = m_dDataSource[i];
    }
    dSum = 0;
    m_dDataResult[0] = 0;
    for (inti = 1; i < DATA_SIZE; i++)
    {
        dSum += m_dDataSource[i] * dDelta;
        m_dDataResult[i] = dSum;
        if (Ymin > m_dDataResult[i]) Ymin = m_dDataResult[i];
        if (Ymax <m_dDataResult[i]) Ymax = m_dDataResult[i];
    }
    m_dInt = dSum;
    UpdateData(false);
    Draw();
}
```

Як бачите відмінність у всіх обробниках тільки в одному рядку при формуванні вихідної функції:

```
m_dDataSource[i] = …;
```

Зверніть також увагу на наступну ділянку коду:

```
	dSum = 0;
	m_dDataResult[0] = 0;
	for (inti = 1; i < DATA_SIZE; i++)
	{
		dSum += m_dDataSource[i] * dDelta;
		m_dDataResult[i] = dSum;
	}
```

Згадайте рис. 1.4. У нульовій точці значення інтеграла дорівнює 0, а кожній наступній зміна значення інтеграла визначається площею прямокутника, побудованого вліво від поточної точки.

Далі необхідно реалізувати функціюDraw() для малювання графіків:

```
void CIntegralDlg::Draw()
{
	int i, nY, nY1;
	CWnd* pWnd = GetDlgItem(IDC_DRAW_AREA);
	CClientDC dc(pWnd);
	CRect rect;
	// Отримуємо розміри вікна для малювання
	pWnd->GetClientRect(&rect);

	int nWidth = rect.right – rect.left + 1;
	int nHeight = rect.bottom – rect.top + 1;
	int x0 = 5;
	int y0 = 5;

	SetDlgItemInt(IDC_Y, Ymax);
	//розраховуємо масштабні коефіцієнти
	double dScaleX = double(nWidth - 10) / ( DATA_SIZE - 1);
	double dScaleY = (nHeight - 10) / (Ymax - Ymin);
	CBrush erazeBr(::GetSysColor(COLOR_MENU));
	// зафарбовуємо область малювання
	dc.FillRect(&rect, &erazeBr);
	// створюємо перо для малювання рамки навколо графіка
	CPen* pPen = new CPen(PS_SOLID, 1, RGB(0, 0, 0));
	dc.SelectObject(pPen);
	dc.MoveTo(0, 0);
	dc.LineTo(nWidth - 1, 0);
	dc.LineTo(nWidth - 1, nHeight - 1);
	dc.LineTo(0, nHeight - 1);
	dc.LineTo(0, 0);
	delete pPen; // видаляємо перо
	// створюємо перо для малювання графіка функції
	pPen =new CPen(PS_SOLID, 2, RGB(0, 0, 255));
	dc.SelectObject(pPen);
	//розраховуємо координату першої точки
	nY =int(nHeight - y0 - (m_dDataSource[0] - Ymin) * dScaleY);
	dc.MoveTo( x0, nY);
	// Малювання графіка вихідної функції
	for (i = 0; i < DATA_SIZE; i++)
	{
		nY =int(nHeight - y0 - (m_dDataSource[i] - Ymin) * dScaleY);
		if ((0 < nY) && (nY < nHeight))
		{
			dc.LineTo(x0 + i * dScaleX, nY);
		}
	}
	delete pPen; // видаляємо перо
	// створюємо перо для малювання графіка інтегралу
	pPen =new CPen(PS_SOLID, 2, RGB(255, 0, 0));
```

```
	dc.SelectObject(pPen);

	nY1 = int(nHeight - y0 - (m_dDataResult[0] - Ymin) * dScaleY);
	dc.MoveTo(x0, nY1);

	// Малювання графіка інтеграла
	for (i = 0; i < DATA_SIZE; i++)
	{
		nY1 = int(nHeight - y0 - (m_dDataResult[i] - Ymin) * dScaleY);
		if ((0 < nY1) && (nY1 < nHeight))
		{
			dc.LineTo(x0 + i * dScaleX, nY1);
		}
	}
	delete pPen; // видаляємо перо

	// Малювання легенди
	pWnd = GetDlgItem(IDC_RESULT);
	CClientDC dc1(pWnd);
	// надаємо фону під текст колір діалогового вікна
	dc1.SetBkColor(RGB(235, 230, 230));

	// створюємо перо для малювання графіка легенди
	pPen =new CPen(PS_SOLID, 2, RGB(255, 0, 0));
	dc1.SelectObject(pPen);
	dc1.MoveTo(0, 10);  // малюємо лінію, колір якої відповідає
	dc1.LineTo(30, 10); // кольору лінії, якою намальований графік функції
	DC1.TextOut(35, 4, _T(" Невизначений інтеграл від y = f(x)"));    // Висновок
тексту

	pPen =new CPen(PS_SOLID, 2, RGB(0, 0, 255));
	dc1.SelectObject(pPen);

	dc1.MoveTo(0, 30);  // малюємо лінію, колір якої відповідає
	dc1.LineTo(30, 30); // кольору лінії, якою намальований графік інтеграла
	DC1.TextOut(35, 20, _T(" Функція y = f(x) ")); // Висновок тексту
	delete pPen; // видаляємо перо
}
```

Загалом нагадаємо алгоритм малювання. Елементи керуванняPictureвикористовуються як про місцедержателей (placeholder). Вони дозволяють прив'язатися до конкретної області екрана. Типова процедура складається з наступних кроків. Отримати вказівникCWndна елемент, створити сумісний контекст пристрою, отримати розміри вікна. Маючи розміри та контекст, ми можемо переходити до малювання.

Щоб графіки заповнювали всю область для малювання необхідно розрахувати масштабні коефіцієнти по осях X і Y.

Далі створюються кисті CBrush якими зафарбовується фон, а також різні пір'я CPen, якими малюватиметься все зображення.

Малювання здійснюється, як правило, у порядку: осі, графіки, підписи даних. Тут треба розуміти, що інформація, що виводиться пізніше, виводиться поверх інформації, що виводиться раніше. Змінюючи порядок виведення елементів, ви можете домогтися того, що осі, наприклад, будуть виводитися поверх графіків.

Оскільки в нашій програмі використовуються математичні функції для успішної компіляції, необхідно підключити бібліотеку.<math.h>.

Після компіляції програма готова до роботи.

VI. Завдання з варіантів

№ варіанту	Лінійна функція		Квадратична функція		Експонентна функція	
	k	*b*	*a*	*b*	*k*	*b*
1	1	1	1	1	1	1
2	2	1	2	1	2	1
3	3	1	3	1	3	1
4	4	1	4	1	4	1
5	1	2	1	2	1	2
6	2	2	2	2	2	2
7	3	2	3	2	3	2
8	4	2	4	2	4	2
9	1	3	1	3	1	3
10	2	3	2	3	2	3
11	3	3	3	3	3	3
12	4	3	4	3	4	3
13	1	4	1	4	1	4
14	2	4	2	4	2	4
15	3	4	3	4	3	4
16	4	4	4	4	4	4
17	1	5	1	5	1	5
18	2	5	2	5	2	5

Індивідуальне завдання

Реалізувати метод правих прямокутників

Контрольні питання

Що таке інтеграл?
У чому різниця між певним та невизначеним інтегралом?
У чому полягає чисельне інтегрування?
Наведіть приклади методів чисельного інтегрування

ЛАБОРАТОРНА РОБОТА №2

Мета роботи – відновити навички роботи з типовими елементами інтерфейсів користувача, динамічними масивами, класами, структурами та функціями на прикладі вирішення завдання чисельного диференціювання.

Завдання:
1) Вивчити теоретичні відомості.
2) Створити базову версію програми чисельного диференціювання.
3) реалізувати індивідуальне завдання.

Теоретичні відомості

За визначенням похідна це межа відношення збільшення функції Δy до збільшення аргументу Δx, при Δx що прагне до нуля.

$$y' = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

Загальний вид розрахунку похідної статечної функції має вигляд:

$$y' = (x^n)' = n * x^{n-1}$$

Для функції $y = x^2$:

$$y' = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{y(x+\Delta x) - y(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{(x+\Delta x)^2 - x^2}{\Delta x} = \lim_{\Delta x \to 0} \frac{x^2 + 2 * x * \Delta x + \Delta x^2 - x^2}{\Delta x} =$$

$$= \lim_{\Delta x \to 0} \frac{2 * x * \Delta x + \Delta x^2}{\Delta x} = \lim_{\Delta x \to 0} (2 * x + \Delta x) = 2 * x$$

У будь-якому математичному довіднику можна знайти таблицю похідних типових функцій.

Крім математичного сенсу похідна також має фізичний сенс - швидкість зміни функції, і геометричний - тангенс кута нахилу, що стосується графіку функції в заданій точці.

Як очевидно з наведеного вище прикладу, алгоритм розрахунку похідної табличної функції дуже простий.

Крок 1. Вибрати точку, де буде визначатися значення похідної.

Крок 2. Використовуючи сусідні точки ліворуч або праворуч, визначити збільшення аргументу.

Крок 3. За відповідними точками визначити збільшення функції.

Крок 4. Знайти відношення збільшення функції до збільшення аргументу.

Як і у випадку з чисельним інтегруванням, для розрахунку похідної можна використовувати сусідні точки як справа, так і зліва. Відповідні методи називаються методом лівих та правих різниць. Пояснимо на прикладі.

Таблиця № 2.1 - Таблична функція y = x2

x_i	0	1	2	3
y_i	0	1	4	9

Допустимо необхідно знайти похідну функції, задану в табл. 2.1 у точці $x_i = 2$. Для цього можна використовувати як точку $x_i = 1$, так і точку $x_i = 3$. Розглянемо обидва випадки.

Метод лівих різниць дає такий результат. Приріст аргументу $\Delta x = 2 - 1 = 1$. Збільшення функції $\Delta y = 4 - 1 = 3$. Значення похідної у точці $y'(2) = \Delta y / \Delta x = 3/1 = 3$.

Використовуючи спосіб правих різниць, отримаємо наступний результат. Приріст аргументу $\Delta x = 3 - 2 = 1$. Збільшення функції $\Delta y = 9 - 4 = 5$. Значення похідної у точці $y'(2) = \Delta y / \Delta x = 5/1 = 5$.

Точне значення похідної у заданій точці $f'(2) = 2x = 2 \cdot 2 = 4$.

Як і у випадку з чисельним інтегруванням, застосування методів лівих або правих різниць може дати похибку як у більшу, так і меншу сторони. Для зменшення похибки також можна зменшувати крок, з яким табульована функція.

Також зверніть увагу на те, що при розрахунку похідної, залежно від використовуваного методу, значення похідної у початковій або кінцевій точці розрахувати неможливо. Тому, якщо вихідна таблична функція має n точок, її похідна буде визначена в n – 1 точках.

Програмна частина

Постановка задачі

Перш ніж приступити до програмної реалізації даного проекту необхідно обумовити вихідні дані, умови та обмеження та ін.

1. Приймемо, що вихідні функції можуть бути визначені будь-якому інтервалі в межах [-10, +10].

2. Кількість точок у табличній функції може бути будь-яким у межах [3, 50].

3. Програма має самостійно формувати вихідну табличну функцію на основі обраної оператором функціональної залежності.

4. Оператор повинен мати можливість вибирати спосіб розрахунку похідної (лівих або правих різниць).

5. Інтерфейс користувача повинен дозволяти оператору задавати межу на якій визначено функцію, задавати кількість точок, вибирати вид функціональної залежності. Також мають виводитися графіки вихідної функції, її аналітичної та чисельної похідної. Дані графіків мають дублюватися у табличному вигляді.

6. Реалізація проекту має враховувати мету лабораторної роботи, тобто. повинні використовуватися класи, структури, динамічна пам'ять, а також різні типові елементи управління на інтерфейсі користувача.

Також слід розуміти, що багато моментів у цій роботі буде реалізовано виключно з навчальної точки зору. Так, наприклад, якщо ми знаємо аналітичні формули для вихідної функції та її похідної, то нам навряд чи коли-небудь буде потрібно чисельне диференціювання цієї функції. Графіки аналітичної та чисельної похідних введені в роботу виключно для порівняння між собою та аналізу впливу на вирішення величини кроку Δx .

Користувальницький інтерфейс

За час навчання програмуванню, ви вже неодноразово стикалися з прикладами того, як одні й ті самі завдання можуть вирішуватися досконало по ращному. У разі також існує безліч способів реалізації вихідних вимог. Так, наприклад, для завдання меж та кількості точок можна використовувати елементиEdit Control, Spin Control, Slider Control. Табличні також можна уявити різними способами (List Box, List Control).

У цій роботі були обрані елементи Slider Control і List Control. Можливість вибору виду функціональної залежності була реалізована з використанням елемента Radio Button.

На рис. 2.1 представлений зовнішній вигляд розробленого інтерфейсу користувача. Пояснимо його структуру.

Традиційно для малювання використовуємо елемент Picture з ідентифікатором IDC_DRAW_AREA. Для індикації мінімальних та максимальних значень за відповідними

осями використано 4 елементиStatic Text з ідентифікаторами IDC_X_MAX, IDC_X_MIN, IDC_Y_MAX і IDC_Y_MIN.

3 елементи Group Box («Межі», «Кількість точок» та «Вихідні функції») логічно відокремлюють різні групи елементів.

Комбінації елементів Static Text і Slider Control формують механізми управління межами диференціювання та кількістю точок. ЕлементиStatic Text праворуч мають ідентифікатори IDC_VALUE_A, IDC_VALUE_B і IDC_VALUE_N. Вони будуть використані для показу поточного значення відповідного параметра.

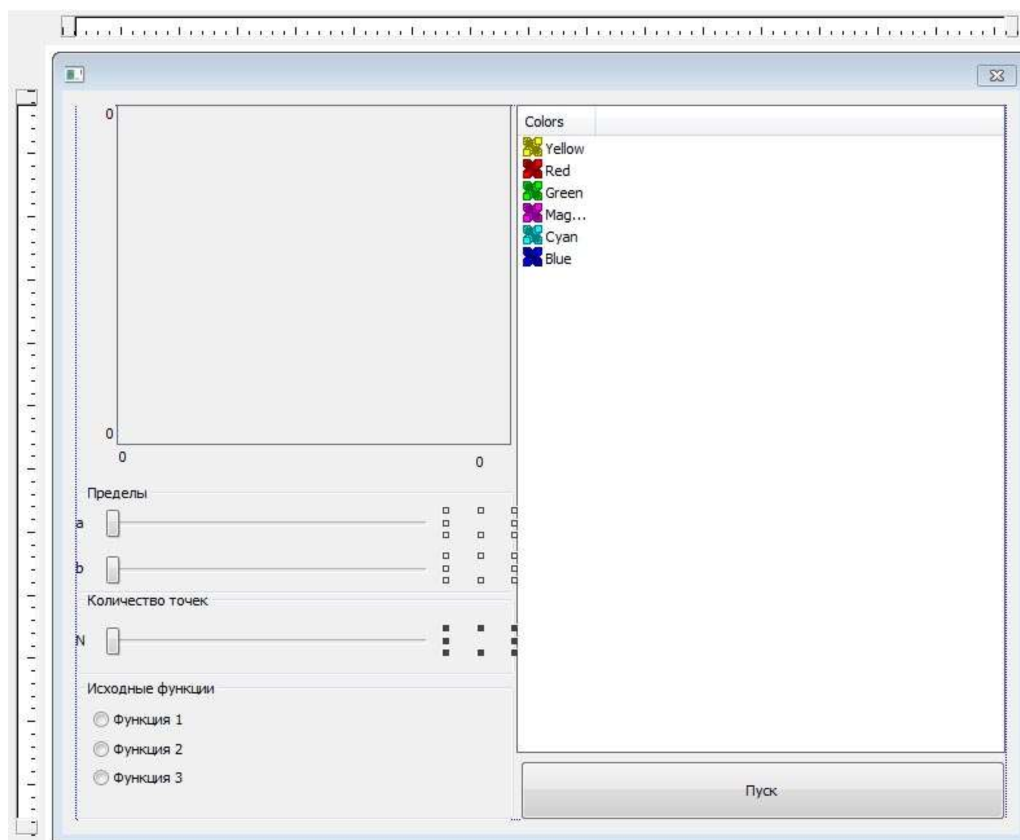


Рисунок 2.1 – Зовнішній вигляд інтерфейсу користувача

Механізм вибору типу функціональної залежності реалізований з використанням елементів Radio Button. Ідентифікатор першої кнопкиIDC_RB_1. Не забудьте також для цієї кнопки виставити на значенняTrue властивості Group.

Праву частину інтерфейсу займають елементи List Control з ідентифікатором IDC_LIST_DATA та звичайна кнопка з ідентифікатором IDC_BTN_RUN.

Для зручності роботи до деяких елементів управління необхідно прив'язати змінні

Ідентифікатор	**Тип**	**Ім'я змінної**
IDC_LIST_DATA	CListCtrl	m_lstData
IDC_RB_1	int	m_FuncType
IDC_SL_A	CSliderCtrl	m_ctrlA
IDC_SL_B	CSliderCtrl	m_ ctrlB
IDC_SL_N	CSliderCtrl	m_ ctrlN

На цьому розробку форми інтерфейсу користувача можна вважати завершеною. Переходимо безпосередньо до програмної реалізації.

Реалізація математичної моделі

Займемося тепер написанням класу, назвемо його DIV, В якому будуть реалізовані всі механізми створення вихідних даних та обчислення похідної.

Почнемо з додавання до проекту класу DIV. Ви можете зробити це, вибравши меню відповідну опцію, або додавши окремоh і cpp файли. У будь-якому випадку, у вас повинні з'явитися файлиdiv.h і div.cpp.

Модифікуйте файлdiv.h наступним чином:

```
#pragma once
#ifndef __DIV_H
#define __DIV_H

// структура зберігання даних однієї точки
struct DivPoint
{
	double X;		// аргумент
	double Y;		// значення функції
	double DivA; // значення аналітичної похідної
	double DivN; // значення чисельної похідної
};

// структура, що об'єднує параметри, необхідні для ініціалізації
struct CreateData
{
	double a;				// Початок інтервалу диференціювання
	double b;				// кінець інтервалу диференціювання
	int Count;				// кількість точок
	double (*pFunction)(double);	// покажчик на вихідну функцію
	double (*pFunctionDiv)(double);	// покажчик на функцію аналітичної похідної
};

// Список методів диференціювання
enum DivType
{
	Left = 0,
	Right = 1
};

// Основний клас
class Div
{
public:
	Div::Div();
	Div::~Div();

	void CreateSourceData(CreateData& cd); // функція створення та ініціалізації
вихідних даних
	void CalcDerivative( DivType method = Left ); // функція розрахунку

	DivPoint*	m_pData;		// покажчик на масив вихідних даних
	int		m_nDataSize;		// розмір цього масиву
	DivType		m_Method;		//Метод диференціювання
};

#endif
```

Пояснимо наведений код.

За умовою завдання необхідно зберігати значення вихідної функції, значення її аналітичної похідної, і навіть значення чисельної похідної. Все це можна зберігати як в окремих масивах, так і в одному. Більш того, для зручності можна об'єднати всі дані в рамках однієї структури та зберігати масив таких структур. У нашому випадку це буде структураDivPoint.

Оскільки нам не відомо заздалегідь кількість точок функції, що диференціюється, нам доведеться виділяти пам'ять для даних динамічно. Для цього нам потрібна низка параметрів, які ми могли б передати у функціюCreateSourceData()безпосередньо, а можемо також поєднати всі ці параметри в рамках однієї структури CreateData.

Швидше за все, тут ви вперше зіткнетеся з вказівниками на функції.

```
double (*pFunction)(double);     // покажчик на вихідну функцію
double (*pFunctionDiv)(double);  // покажчик на функцію аналітичної похідної
```

Хоча використання тут даного механізму не зовсім виправдане (показаний з навчальної точки зору), він дозволяє суттєво розширити можливості вашої програми.

З точки, функції CreateSourceData()їй необхідно знати функцію, що описує вихідні дані та її похідну. Це цілком можна реалізувати визначивши дві відповідні функції у класіDIV. Але як бути, якщо ви хочете досліджувати інші функціональні залежності? Весь час колупатися в коді у пошуках потрібного місця та змінювати математику? Чому б не винести всі можливі функції в окремий файл та не викликати потрібну функцію в момент створення вихідних даних? Саме для цього і додано наші два вказівники на функції.

Більш практичний приклад використання покажчиків на функції ви можете вивчити за посиланням: https://ravesli.com/urok-104-ukazateli-na-funktsii/. Коротко допустимо ви пишете свою функцію сортування. Усі сортування зводяться до того що, що порівнюються два значення і за результатами порівняння або змінюються місцями, чи ні. Допустимо ви хочете додати у свою функцію можливість сортувати елементи в порядку зростання або зменшення. Вам достатньо визначити дві різні функції порівняння, а в основному алгоритмі використовувати покажчик на функцію порівняння. Тоді при безпосередньому виклику функції сортування ви зможете підставити як аргумент необхідну функцію порівняння та отримати різні результати.

Для полегшення можливості вибору методу диференціювання введений тип даних, що перераховуєтьсяDivType.

Власне, клас DIVПояснень не потребує. Звичайна схема: конструктор, деструктор, функція створення, функція розрахунку, змінні зберігання поточних параметрів.

Тепер модифікуємо файл div.cpp

```
#include "pch.h"
#include "Div.h"

Div::Div()
{
	m_pData = NULL;
	m_nDataSize = 0;
}

Div::~Div()
{
	if (m_pData) delete m_pData;
}

void Div::CreateSourceData(CreateData& cd)
{
	// якщо раніше пам'ять було виділено, звільняємо її
	if (m_pData) delete m_pData;
```

```
        // виділяємо пам'ять під нову кількість точок
        m_pData = new DivPoint[cd.Count];
        // обнулюємо всю виділену пам'ять
        memset(m_pData, 0,sizeof(DivPoint) * cd. Count);
        m_nDataSize = cd.Count;
        // обчислюємо крок розбиття інтервалу
        double dx = (cd.b - cd.a) / (cd.Count - 1);
        double x = cd.a;
        // перебираємо всі точки інтервалу
        for (inti = 0; i <cd.Count; i++)
        {
                // формуємо вихідні дані
                m_pData[i].X = x;
                m_pData[i].Y = cd.pFunction(x);
                m_pData[i].DivA = cd.pFunctionDiv(x);

                x + = dx;
        }
}

void Div::CalcDerivative( DivType method )
{
        // запам'ятовуємо вибраний метод
        m_Method = method;
        switch (method)
        {
                case Left:
                {
                        for (inti = 1; i < m_nDataSize; i++)
                        {
                                m_pData[i].DivN = (m_pData[i].Y - m_pData[i - 1].Y) /
(m_pData[i].X - m_pData[i - 1].X);
                        }
                }
                break;
                case Right:
                {
                }
                break;
                default: break;
        }
}
```

Оскільки ми працюємо з динамічною пам'яттю, необхідно перед початком роботи проініціалізувати покажчики, а після завершення роботи звільнити виділену раніше пам'ять. Що і зроблено в конструкторі та деструкторі.

За умовою завдання користувач може самостійно змінювати кількість точок, куди розбивається інтервал диференціювання. Отже, при кожному виклику функціїCreateSourceData() необхідно перевіряти покажчик на дані m_pData. Якщо він не порожній, значить раніше вже виділялася пам'ять і її необхідно очистити.

Пам'ятайте, як у структурі CreateData ми оголосили два вказівники на функції?

```
        double (*pFunction)(double);
        double (*pFunctionDiv)(double);
```

Тепер такий підхід дозволяє використовувати при формуванні вихідних даних саме ті функції які вказав користувач. При цьому немає потреби змінювати програмний код.

```
        m_pData[i].Y =cd.pFunction(x);
```

```
m_pData[i].DivA = cd.pFunctionDiv(x);
```

Алгоритм роботи функції CalcDerivative()реалізований лише для методу лівих різниць. Реалізацією методу правих різниць вам необхідно буде зайнятися у рамках індивідуального завдання.

Нагадай, що з методу лівих різниць неможливо розрахувати значення похідної нульової точки інтервалу, т.к. для неї не існує точки зліва на підставі якої розраховується різниця. Тому цикл починається не з нульової, а першої точки. Для способу правих різниць ситуація протилежна.

Тепер давайте визначимось із набором функцій, які користувач може аналізувати. Для зручності їх можна описати в одному h-файлі. Додайте файл файлfunctions.h і модифікуйте його в такий спосіб.

```
#pragma once
#ifndef __FUNCTIONS_H
#define __FUNCTIONS_H

#include <math.h>

double PowFunction(double x) {return x * x; }
double ExpFunction(double x) { return exp(x); }
double SinFunction(double x) { return sin(x); }

double PowFunctionDiv(double x) { return 2* x; }
double ExpFunctionDiv(double x) { return exp(x); }
double SinFunctionDiv(double x) { return cos(x); }

#endif
```

Як бачите ми визначили тут три оригінальні функції та три відповідні їм аналітичні похідні.

У цьому математичну модель вважатимуться завершеною.

<u>Інтеграція інтерфейсу та математичної моделі</u>

Тепер необхідно об'єднати наш інтерфейс користувача з математичною моделлю і реалізувати механізми візуалізації даних.

Почнемо з модифікації заголовного файлу DerivativeDlg.h.

Необхідно підключити файл Div.h та зробити необхідні оголошення.

```
// DerivativeDlg.h: файл заголовка
//

#pragma once

#include "Div.h"

// Діалогове вікно CDerivativeDlg
class CDerivativeDlg : public CDialogEx
{
// Створення
public:
	CDerivativeDlg(CWnd* pParent = nullptr);	// Стандартний конструктор

	Div m_Derivative;   // змінна класу DIV
	void Draw();        // функція малювання графіків
	void UpdateTable(); // функція заповнення таблиці
```

Всі подальші зміни стосуються файлу DerivativeDlg.cpp. Насамперед необхідно підключити стандартну бібліотекуstring і наш файл functions.h

```
#include <string>
#include "functions.h"
```

Бібліотека string знадобиться нам для перетворення текстової інформації, що виводиться в інтерфейс.

Традиційно наприкінці функції OnInitDialog() додаємо початкову ініціалізацію все, що цього вимагає.

```
	// TODO: додайте додаткову ініціалізацію
	// встановлення діапазонів та поточного положення для елементів Slider
	m_ctrlA.SetRange(0, 100);
	m_ctrlA.SetPos(50);
	m_ctrlB.SetRange(0, 100);
	m_ctrlB.SetPos(60);
	m_ctrlN.SetRange(3, 50);
	m_ctrlN.SetPos(3);

	// ініціалізація текстових значень, що відповідають положенням елементів Slider
	double dValue;
	CString str;
	dValue = -10 + 50 * 20/100.;
	str.Format(_T("%3.3f"), dValue);
	SetDlgItemText(IDC_A_VALUE, str);
	dValue = -10 + 60 * 20/100.;
	str.Format(_T("%3.3f"), dValue);
	SetDlgItemText(IDC_B_VALUE, str);
	str.Format(_T("%d"), 3);
	SetDlgItemText(IDC_N_VALUE, str);

	// формування шпальт в елементі List Control
	m_lstData.InsertColumn(0,_T("X"), 0, 60);
	m_lstData.InsertColumn(1,_T("Y"), 0, 60);
	m_lstData.InsertColumn(2,_T("Аналітична"), 0, 100);
	m_lstData.InsertColumn(2,_T("Чисельна"), 0, 100);

	// задаємо початкове значення змінної, що визначає обрану функцію m_FuncType = 0;

	// ініціюємо оновлення та перемальовування інтерфейсу користувача
	UpdateData(false);
```

При аналізі коду ініціалізації текстових значень пригадайте, що за умовою завдання необхідно задавати інтервал диференціювання в діапазоні [-10;10]. Саме для реалізації цієї вимоги значення повзунків на елементахSlider перетворюються так:

```
	dValue = -10 + 50 * 20 / 100.;
```

де:

- -10 - Початок інтервалу;
- 50 - поточне значення величини;
- 20 - Довжина інтервалу;
- 100 - максимальне значення величини.

Далі реалізуємо обробник подій для елементів Slider.

Додати обробник системної події WM_HSCROL модифікуйте його в такий спосіб.

```
void CDerivativeDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	// TODO: додайте свій код обробника повідомлень або виклик стандартного

	double dValue;
	CString str;
	if (pScrollBar == GetDlgItem(IDC_SL_A))
	{
		// Ліва межа інтервалу не може бути більшою або рівною правою
		if (m_ctrlA.GetPos() >= m_ctrlB.GetPos())
		{
			m_ctrlA.SetPos(m_ctrlB.GetPos() - 1);
		}
		// виводимо відповідне текстове значення
		dValue = -10 + m_ctrlA.GetPos() * 20 / 100.;
		str.Format(_T("%3.3f"), dValue);
		SetDlgItemText(IDC_A_VALUE, str);
	}
	else if (pScrollBar == GetDlgItem(IDC_SL_B))
	{
		// Права межа інтервалу може бути менше чи рівної лівої
		if (m_ctrlB.GetPos() <= m_ctrlA.GetPos())
		{
			m_ctrlB.SetPos(m_ctrlA.GetPos() + 1);
		}
		// виводимо відповідне текстове значення
		dValue = -10 + m_ctrlB.GetPos() * 20 / 100.;
		str.Format(_T("%3.3f"), dValue);
		SetDlgItemText(IDC_B_VALUE, str);
	}
	else if (pScrollBar == GetDlgItem(IDC_SL_N))
	{
		str.Format(_T("%d"), m_ctrlN.GetPos());
		SetDlgItemText(IDC_N_VALUE, str);
	}

	CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

На даному етапі (якщо ви створите порожні обробники-«заглушки» для функційDraw() і UpdateTable()) Ви можете скомпілювати та запустити програму та переконатися, що значення елементів Slider виводяться правильно і всі обмеження працюю коректно.

Далі реалізуємо обробник натискання на кнопку "Пуск".

```
void CDerivativeDlg::OnBnClickedBtnRun()
{
	// TODO: додайте свій код обробника повідомлень
	UpdateData( true );

	CreateData cd;
	// отримуємо поточні значення меж інтервалу диференціювання
	cd.a = -10 + m_ctrlA.GetPos() * 20/100.;
	cd.b = -10 + m_ctrlB.GetPos() * 20/100.;
	cd.Count = m_ctrlN.GetPos();
	// ініціалізуємо вказівники на необхідні функції
	switch (m_FuncType)
	{
		case 0:
		default:
		{
			cd.pFunction = PowFunction;
			cd.pFunctionDiv = PowFunctionDiv;
```

```
		}
		break;
		case 1:
		{
			cd.pFunction = ExpFunction;
			cd.pFunctionDiv = ExpFunctionDiv;
		}
		break;
		case 2:
		{
			cd.pFunction = SinFunction;
			cd.pFunctionDiv = SinFunctionDiv;
		}
		break;
	}
	// викликаємо функцію створення вихідних даних
	m_Derivative.CreateSourceData(cd);
	// Викликаємо функцію диференціювання
	m_Derivative.CalcDerivative();
	// оновлюємо вміст таблиці
	UpdateTable();
	// малюємо графіки
	Draw();
}
```

Тепер займемося реалізацією найоб'ємнішої функції нашого проекту, а саме функцією Draw().

```
void CDerivativeDlg::Draw()
{
	if (!m_Derivative.m_pData) return;
	CWnd* pWnd = GetDlgItem(IDC_DRAW_AREA);
	CClientDC dc(pWnd);
	CRect rect;
	// Отримуємо розміри вікна для малювання
	pWnd->GetClientRect(&rect);

	int nWidth = rect.right - rect.left + 1;
	int nHeight = rect.bottom - rect.top + 1;
	int nBorder = 5;
	int i;
	// визначаємо мінімальні та максимальні значення по шпалерах осях
	double dYmin = m_Derivative.m_pData[0].Y;
	double dYmax = m_Derivative.m_pData[0].Y;
	for (i = 0; i < m_Derivative.m_nDataSize; i++)
	{
		if (dYmin > m_Derivative.m_pData[i].Y) dYmin = m_Derivative.m_pData[i].Y;
		if (dYmax <m_Derivative.m_pData[i].Y) dYmax = m_Derivative.m_pData[i].Y;
		if (dYmin > m_Derivative.m_pData[i].DivA) dYmin = 
m_Derivative.m_pData[i].DivA;
		if (dYmax <m_Derivative.m_pData[i].DivA) dYmax = 
m_Derivative.m_pData[i].DivA;
		if (dYmin > m_Derivative.m_pData[i].DivN) dYmin = 
m_Derivative.m_pData[i].DivN;
		if (dYmax <m_Derivative.m_pData[i].DivN) dYmax = 
m_Derivative.m_pData[i].DivN;
	}

	SetDlgItemInt(IDC_X_MIN, int(m_Derivative.m_pData[0].X));
	SetDlgItemInt(IDC_X_MAX, int(m_Derivative.m_pData[m_Derivative.m_nDataSize - 
1].X));
```

```
	SetDlgItemInt(IDC_Y_MIN, int(dYmin));
	SetDlgItemInt(IDC_Y_MAX, int(dYmax));

	//розраховуємо масштабні коефіцієнти
	double dScaleX = double(nWidth - 2 * nBorder) /
(m_Derivative.m_pData[m_Derivative.m_nDataSize - 1].X - m_Derivative.m_pData[0].X);
	double dScaleY = (nHeight - 2 * nBorder) / (dYmax - dYmin);
	CBrush erazeBr(::GetSysColor(COLOR_MENU));// зафарбовуємо область малювання
	dc.FillRect(&rect, &erazeBr);
	// створюємо перо для малювання рамки навколо графіка
	CPen* pPen = new CPen(PS_SOLID, 1, RGB(0, 0, 0));
	dc.SelectObject(pPen);
	dc.MoveTo(0, 0);
	dc.LineTo(nWidth - 1, 0);
	dc.LineTo(nWidth - 1, nHeight - 1);
	dc.LineTo(0, nHeight - 1);
	dc.LineTo(0, 0);
	delete pPen; // видаляємо перо

	int nY;

	// створюємо перо для малювання графіка аналітичної похідної
	pPen =new CPen(PS_SOLID, 3, RGB(255, 0, 0));
	dc.SelectObject(pPen);

	//розраховуємо координату першої точки
	nY =int(nHeight - nBorder - (m_Derivative.m_pData[0].DivA - dYmin) * dScaleY);
	dc.MoveTo(nBorder, nY);

	// Малювання графіка
	for (i = 0; i < m_Derivative.m_nDataSize; i++)
	{
		nY =int(nHeight - nBorder - (m_Derivative.m_pData[i].DivA - dYmin) *
dScaleY);
		if ((0 < nY) && (nY < nHeight))
		{
			dc.LineTo(nBorder + (m_Derivative.m_pData[i].X -
m_Derivative.m_pData[0].X) * dScaleX, nY);
		}
	}
	delete pPen; // видаляємо перо

	// створюємо перо для малювання графіка чисельної похідної
	pPen =new CPen(PS_SOLID, 2, RGB(0, 0, 255));
	dc.SelectObject(pPen);

	if (m_Derivative.m_Method == Left)
	{
		//розраховуємо координату першої точки
		nY =int(nHeight - nBorder - (m_Derivative.m_pData[1].DivN - dYmin) *
dScaleY);
		dc.MoveTo(nBorder + (m_Derivative.m_pData[1].X -
m_Derivative.m_pData[0].X) * dScaleX, nY);
		// Малювання графіка
		for (i = 1; i < m_Derivative.m_nDataSize; i++)
		{
			nY =int(nHeight - nBorder - (m_Derivative.m_pData[i].DivN - dYmin) *
dScaleY);
			if ((0 < nY) && (nY < nHeight))
			{
				dc.LineTo(nBorder + (m_Derivative.m_pData[i].X -
m_Derivative.m_pData[0].X) * dScaleX, nY);
```

```
				}
			}
		}
		else
		{
			//розраховуємо координату першої точки
			nY =int(nHeight - nBorder - (m_Derivative.m_pData[0].DivN - dYmin) *
dScaleY);
			dc.MoveTo(nBorder, nY);
			// Малювання графіка
			for (i = 0; i < m_Derivative.m_nDataSize - 1; i++)
			{
				nY =int(nHeight - nBorder - (m_Derivative.m_pData[i].DivN - dYmin) *
dScaleY);
				if ((0 < nY) && (nY < nHeight))
				{
					dc.LineTo(nBorder + (m_Derivative.m_pData[i].X -
m_Derivative.m_pData[0].X) * dScaleX, nY);
				}
			}
		}
		delete pPen; // видаляємо перо

		// створюємо перо для малювання графіка функції
		pPen =new CPen(PS_SOLID, 1, RGB(0, 200, 0));
		dc.SelectObject(pPen);

		//розраховуємо координату першої точки
		nY =int(nHeight - nBorder - (m_Derivative.m_pData[0].Y - dYmin) * dScaleY);
		dc.MoveTo(nBorder, nY);
		// Малювання графіка вихідної функції
		for (i = 0; i < m_Derivative.m_nDataSize; i++)
		{
			nY =int(nHeight - nBorder - (m_Derivative.m_pData[i].Y - dYmin) *
dScaleY);
			if ((0 < nY) && (nY < nHeight))
			{
				dc.LineTo(nBorder + (m_Derivative.m_pData[i].X -
m_Derivative.m_pData[0].X) * dScaleX, nY);
			}
		}
		delete pPen; // видаляємо перо
}
```

Базовий алгоритм малювання такий самий, як і в попередній роботі. Але можливості цієї функції значно розширені. Оскільки функція малюватиме графіки трьох функцій (вихідна, аналітична похідна та чисельна похідна), для «краси» (графіки займатимуть усе відведене їм місце) їй необхідні масштабні коефіцієнти для осейX і Y. Отже знадобляться мінімальне та максимальне значення по осяхX і Y розрахунок яких і додано на початку функції.

Для того, щоб графіки не перекривали один одного, було вирішено малювати їх у певному порядку з поступовим зменшенням товщини лінії.

Залишилось реалізувати останню функцію UpdateTable().

```
void CDerivativeDlg::UpdateTable()
{
	m_lstData.DeleteAllItems();
	CString strText;

	for (inti = 0; i < m_Derivative.m_nDataSize; i++)
	{
```

```
			strText.Format(TEXT("%g"), m_Derivative.m_pData[i].X);
			m_lstData.InsertItem(LVIF_TEXT | LVIF_STATE, i, strText, 0, 0, 0, 0);

			strText.Format(TEXT("%g"), m_Derivative.m_pData[i].Y);
			m_lstData.SetItemText(i, 1, strText);

			strText.Format(TEXT("%g"), m_Derivative.m_pData[i].DivA);
			m_lstData.SetItemText(i, 2, strText);

			strText.Format(TEXT("%g"), m_Derivative.m_pData[i].DivN);
			m_lstData.SetItemText(i, 3, strText);
	}

	strText.Format(TEXT("--"));
	if (m_Derivative.m_Method == Left)
			m_lstData.SetItemText(0, 3, strText);
}
```

Алгоритм цієї функції також досить типовий: видаляємо старий вміст; поелементно заповнюємо всі осередки таблиці новими даними. Нюанс тут у тому, що для методу лівих різниць відсутні дані для першого рядка чисельної похідної. А для методу правих – дані останнього рядка. У прикладі реалізовано перевірку для методу лівих різниць:

```
	strText.Format(TEXT("--"));
	if (m_Derivative.m_Method == Left)
			m_lstData.SetItemText(0, 3, strText);
```

Тепер програма повністю готова для компіляції та запуску. Якщо все зроблено правильно, то на екрані ви побачите щось подібне до рис. 2.2. Зелений графік – вихідна функція. Червоний – аналітична похідна. Синій – чисельна похідна.

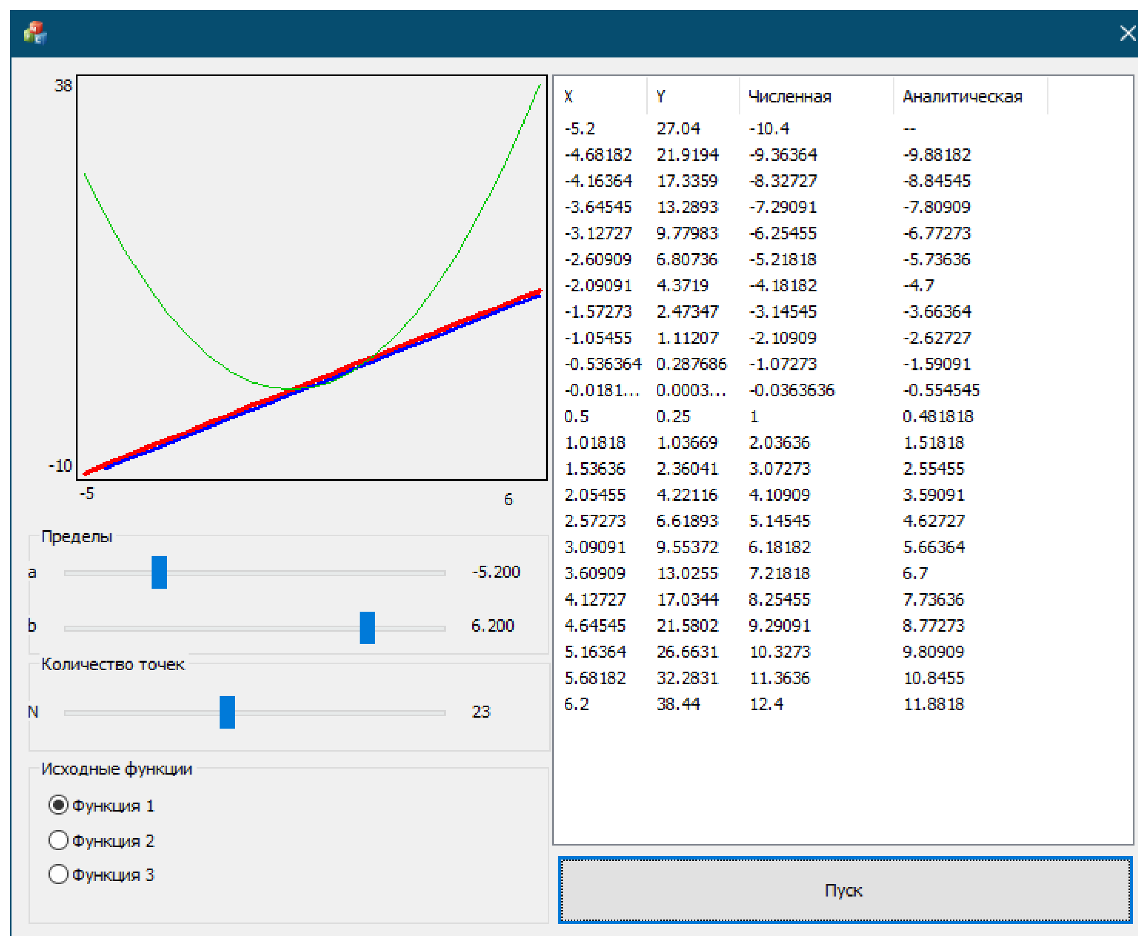


Рисунок 2.2 – Результат роботи програми.

Індивідуальне завдання

В рамках індивідуального завдання необхідно реалізувати:

- можливість вибору на інтерфейсі користувача методу диференціювання (правих або лівих різниць);

- реалізувати метод правих різниць;

- Після реалізації методу правих різниць та налагодження програми, необхідно реалізувати чисельне диференціювання функції відповідно до варіанту. У цьому висновок аналітичної похідно непотрібен т.к. деякі функції досить складні для диференціювання. Зверніть увагу, що в деяких функціях здійснюється розподіл на x, а x може дорівнювати 0.

Завдання з варіантів

№	Функція	№	Функція
1	$y = 2x^2 + 3x$	13	$y = -x^2 + x$
2	$y = 2^x - 1$	14	$y = e^x + 2$
3	$y = 1/x$	15	$y = x + 1/x$
4	$y = x\sin(x)$	16	$y = x\cos(x)$
5	$y = x^3$	17	$y = -x + x^2$
6	$y = 1/2^x$	18	$y = 1/x^2$
7	$y = 2\sin(x+2)$	19	$y = -\cos(2x+1)$
8	$y = x^2 + 0.5x$	20	$y = x^2 + x^3$

9	$y = e^x - x$	21	$y = e^x - x^2$
10	$y = 2/(3x+1)$	22	$y = (x+1)/x^2$
11	$y = \sin(x)/x$	23	$y = \cos(x)/x$
12	$y = e^x / x$	24	$y = x^2/(x+1)$

Контрольні питання

Що таке похідна?
Як обчислити похідну чисельно? Які методи ви знаєте?
Що таке клас?
Що таке структура?
Робота з динамічною пам'яттю у с++.
Робота з функціями користувача в с++.

ЛАБОРАТОРНА РОБОТА №3

Мета роботи - розробити програмну модель автоматизованої системи управління рівнем рідини та провести на ній базові дослідження питань управління.

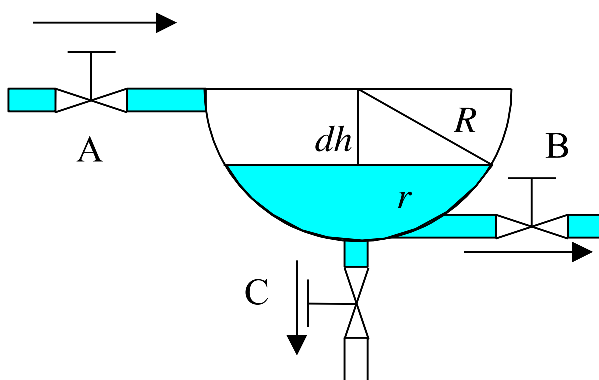
Завдання:

1) Вивчити математичну модель процесу
2) Реалізувати модель використовуючи програмний код із даних методичних вказівок
3) Вивчити роботу моделі

Теоретичні відомості

Опис процесу

Розглядається у роботі фізичний процес дуже нагадує всім відомий зливний бачок. Є ємність напівсферичної форми в яку через трубу А надходить вода, через трубу відкачується, а через трубу С виходить «самотеком» (див. рис. 3.1).



Малюнок 3.1. - Модель процесу

Приймемо, що радіус ємності $R = 0.5$ м.

Керуючи положенням засувок*A* і *B* можна змінювати витрату води, що протікає.

Приймемо, що витрати вхідного та вихідного потоку води можуть змінюватися від 0 до 10 кг/с.

Приймемо, що засувка *З* може бути повністю відкрита, або повністю закрита.

Для підтримки рівня рідини в ємності, на перший погляд, необхідно щоб витрати води на виході та на вході були рівними (із закону збереження мас).

Нехай витік води через трубу *З*імітує різні протікання негерметичних стиків труб тощо. За умови, що зливний отвір закрито, наша система буде підтримувати витрати води на виході, що дорівнює витраті води на вході. Очевидно, що при витоку води через трубу*З*закон збереження мас виконуватися нічого очікувати і має зменшуватися, а сам принцип управління заснований на законі збереження не работает. Вирішення цієї проблеми буде показано пізніше в лабораторній роботі №6.

Математична модель процесу

Розглянемо питання, наскільки зміниться рівень води Δh у ємності, якщо її обсяг зміниться на величину ΔV (Рис. 3.2).

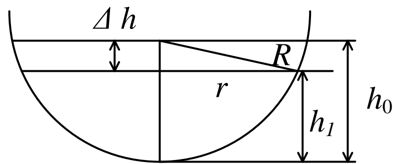


Рисунок 3.2 – До питання визначення рівня води

Для малих значень ΔV можна вважати, що стінки судини є циліндричними. Тоді справедлива формула:

$$\Delta V = \pi \cdot r^2 \cdot \Delta h \quad (1)$$

де $r^2 = R^2 - \Delta h^2$ (за теоремою Піфагора див. рис. 3.2)

або ж:

$$\Delta V = \pi \cdot (R^2 - \Delta h^2) \cdot \Delta h \quad (2)$$

Визначити величину Δh у формулі (2) можна в такий спосіб.
Запишемо (2) у вигляді інтеграла:

$$\Delta V = \int_{h_0}^{h_1} \pi \cdot (R^2 - h^2) dh \quad (3)$$

За умови, що величини ΔV і R відомі, завдання визначення величини Δh зводиться до визначення меж інтегрування. Для цього необхідно поступово збільшувати інтервал [h_0 , h_1] на величину dh доти, доки права частина рівняння (3) не стане рівною лівою. Знаючи межі інтегрування, можна визначити величину Δh як:

$$\Delta h = h_1 - h_0 \quad (4)$$

Очевидно, що менше величина dh, Тим точніше розрахунки. Надалі ми повернемося до питання вибору даної величини, а поки що приймемо, що dh = 1 мм.

Тепер визначимо значення величини ΔV. Це легко зробити на підставі закону збереження маси.

$$\Delta V = \frac{(G_A - G_B) \cdot \tau}{\rho} - V_{ym} \quad (5)$$

де: ρ - Щільність води;

V_{ym} - об'єм води, що витікає через отвір С (див. рис. 3.1);

τ - Час.

Витрати води G_A і G_B на вході та виході ємності відомі т.к. задаються або вручну, або автоматично. Щільність води приймемо $\rho = 1000\,кг/м^3$. Час моделювання τ ми будемо ставити програмно. Залишилось знайти величину $V_{ут}$.

Відповідно, формулі Торрічеллі швидкість потоку рідини через отвір визначається як:

$$\upsilon^2 = 2 \cdot g \cdot H \tag{6}$$

де υ - Швидкість потоку рідини;
g - сила тяжіння
H - Висота рідини над отвором ($H = R - dh$ див. рис. 3.1)

Приймемо радіус зливного отвору r рівним 10 мм. Тоді обсяг води $V_{ут}$ проходить через нього за час τ визначатиметься виразом:

$$V_{ут} = \pi \cdot r^2 \cdot \upsilon \cdot \tau \tag{7}$$

або з урахуванням (6)

$$V_{ут} = \pi \cdot r^2 \cdot \sqrt{2 \cdot g \cdot H} \cdot \tau \tag{8}$$

Завдання управління у разі тривіальна. Будь-якої миті часу:

$$G_B(t) = G_A(t) \tag{9}$$

Таким чином, ми розробили необхідне математичне забезпечення нашої моделі і можемо переходити до програмної реалізації.

Рисунок 3.3 – алгоритм роботи моделі

Програмна частина

Постановка задачі

Давайте спершу визначимося з тим, що ми хочемо отримати.

1) Необхідно реалізувати програмну модель процесу, зображеного на рис. 3.1.
2) Програма має моделювати процес у режимі реального часу.
3) Програма має показувати:
- поточний рівень води в ємності;
- Витрати води на вході та виході.
4) Користувач повинен мати можливість у ручному режимі роботи змінювати витрати води на вході та виході. В автоматичному режимі – лише на вході.
5) В автоматичному режимі витрата води на виході повинна дорівнювати витраті води на вході.
6) Користувач повинен мати можливість вмикати/вимикати автоматичний режим роботи, а також можливість відкривати/закривати зливний отвір у трубі 3 (Див. рис. 3.1)

Користувальницький інтерфейс

Створіть проект SimpleModelASUна основі діалогового класу. Традиційно роботу над програмною частиною проекту почнемо з розробки заготовки інтерфейсу користувача. Відкрийте файлSimpleModelASU.rc2 і відредагуйте форму діалогу оскільки це показано на рис 3.3.

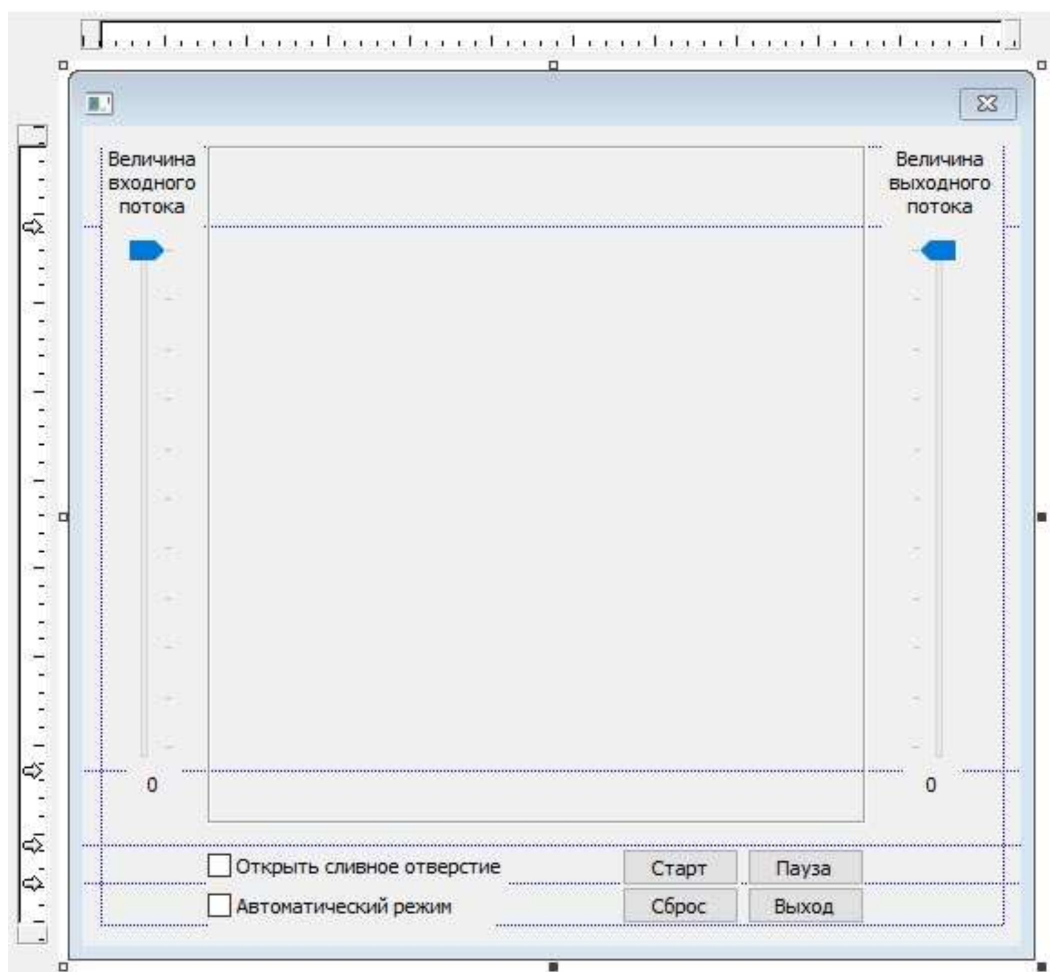


Рисунок 3.3 – Інтерфейс користувача

Не забувайте, що для вирівнювання окремих елементів у формі можна використовувати лінії прив'язки. Для цього достатньо клікнути мишкою на лінійці, що відображається зліва або зверху.

Пояснимо структуру інтерфейсу.

Ліворуч і праворуч присутні два повзунки (Slider) IDC_SLIDER_IN і IDC_SLIDER_OUTвідповідно. Над та під ними елементиStatic Text. Два нижні елементи використовуються для індикації поточного значення витрат. Їхні ідентифікаториIDC_IN_VALUE і IDC_OUT_VALUEвідповідно. Елементам CheckBox дайте ідентифікаториIDC_CHECK_OPEN і IDC_CHECK_AUTO. Перший імітуватиме відкриття/закриття труби*3*, а другий вмикати/вимикати автоматичний режим.

Ідентифікатори кнопокIDC_BTN_START, IDC_BTN_PAUSE, IDC_BTN_RESET, IDC_BTN_EXIT.

У центрі форми розташований елемент Picture Control з ідентифікатором IDC_DRAW.

До деяких елементів управління необхідно прив'язати змінні.

Ідентифікатор	Тип	Ім'я змінної
IDC_CHECK_AUTO	CButton	m_Auto
IDC_CHECK_OPEN	CButton	m_Open
IDC_SLIDER_IN	CSliderCtrl	m_slIn
IDC_SLIDER_OUT	CSliderCtrl	m_slOut

Створюємо обробники кнопок OnBnClickedBtnStart(), OnBnClickedBtnPause(), OnBnClickedBtnReset(), OnBnClickedBtnExit(), OnBnClickedCheckAuto().

Додатково нам знадобляться обробники системних подій OnTimer() (Подія WM_TIMER) таOnVScroll() (Подія WM_VSCROLL).

Реалізація моделі процесу

Для реалізації моделі нам потрібні допоміжні змінні для зберігання таких величин як витрати води на всіх трубах, поточний рівень води і поточний час. Також ми рознесемо за окремими функціями процедуру розрахунку параметрів моделі та процедуру малювання.

Необхідно також пояснити, як ми збираємося реалізувати сам процес моделювання. За умовою завдання він має відбуватися як реального часу. Що це означає? Система працює в режимі реального часу якщо вона встигає обробляти всю інформацію, що надходить до неї. Відновлювати поточний стан нашої моделі ми будемо за сигналами таймера. Інтервалу спрацьовування таймера 100 мс цілком достатньо.

Почнемо з оголошення змінних та функцій. У файліSimpleModelASUDlg.h модифікуйте конструктор діалогового вікна таким чином:

```
// Діалогове вікно CSimpleModelASUDlg
class CSimpleModelASUDlg : public CDialogEx
{
// Створення
public:
	CSimpleModelASUDlg(CWnd* pParent = nullptr);  // Стандартний конструктор

	double m_dInFlowRate; // Витрата води на вході, кг/с
	double m_dOutFlowRate; // Витрата води на виході, кг/с
	double m_dDrainFlowRate; // Витрата води на сливі, кг/с
	double m_dH; // рівень води у резервуарі, м
	double m_dTime; // час моделювання, з
	void Draw(); // функція малювання бака
	void Calc(); // функція розрахунку рівня води у баку
```

Всі подальші зміни робимо у файлі SimpleModelASUDlg.cpp.

Оскільки ми будемо використовувати специфічні математичні операції, необхідно підключити бібліотеку math.h. Також для зручності ми можемо визначити макрос із зрозумілим ім'ям для завдання інтервалу спрацьовування таймера

```
#include <math.h>
#define DELTA_TIME 100 // ms
```

Далі потрібно провести початкову ініціалізацію. Перейдіть до обробникаOnInitDialog() і додайте в його кінець код ініціалізації:

```
	// TODO: додайте додаткову ініціалізацію
	m_slIn.SetRange(0, 100, TRUE);   // задаємо діапазон зміни слайдерів
	m_slOut.SetRange(0, 100, TRUE);
	m_slOut.EnableWindow(FALSE);     // забороняємо змінювати витрати води на виході
	m_Open.SetCheck(0); // зливний отвір відкритий
	m_Auto.SetCheck(1); // увімкнено автоматичний режим
	SetDlgItemText(IDC_IN_VALUE, _T("0")); //Витрата води перебувати на
	SetDlgItemText(IDC_OUT_VALUE, _T("0")); // нульовий позначці
	//Початкові значення
	m_dTime = 0;
	m_dH = 0.5;
	m_dInFlowRate = 0;
	m_dOutFlowRate = 0;
	m_dDrainFlowRate = 0;
```

Нагадуємо, що додаткова ініціалізація у функціїOnInitDialog() додається після коментаря«// TODO: додайте додаткову ініціалізацію». Це зроблено для того, щоб не втручатися в послідовність операцій, які виконуються під час запуску програми.

Далі реалізуємо механізм зміни витрат на вході та виході. Перейдіть до обробникаOnVScroll() і модифікуйте його так:

```
void CSimpleModelASUDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	// TODO: додайте свій код обробника повідомлень або виклик стандартного
	if (m_Auto.GetCheck() == 1)
	{
		CScrollBar* pScroll = (CScrollBar*)GetDlgItem(IDC_SLIDER_IN);
		if (pScroll == pScrollBar)
		{
			m_slOut.SetPos(m_slIn.GetPos()); // Витрата на вході дорівнює 
витрати на виході
		}
	}

	// Розрахунок показання слайдербарів
	m_dInFlowRate = m_slIn.GetPos() * 10. / 100.;
	m_dOutFlowRate = m_slOut.GetPos() * 10. / 100.;

	// Виведення показань слайдербарів на екран
	CString str;
	str.Format(TEXT("%g"), m_dInFlowRate);
	SetDlgItemText(IDC_IN_VALUE, str);

	str.Format(_T("%g"), m_dOutFlowRate);
	SetDlgItemText(IDC_OUT_VALUE, str);

	CDialogEx::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

На цьому етапі можна скомпілювати програму, запустити її і переконатися, що при переміщенні повзунків змінюються відповідні показання.

Тепер модернізуємо обробникOnBnClickedCheckAuto() та заборонимо можливість зміни витрати на виході при включеному автоматичному режимі:

```
void CSimpleModelASUDlg::OnBnClickedCheckAuto()
{
	// TODO: додайте свій код обробника повідомлень
	if (m_Auto.GetCheck() == 1) // увімкнення/вимкнення автоматичного режиму
	{
		m_slOut.EnableWindow(FALSE);
		m_slOut.SetPos(m_slIn.GetPos());
	}
	else
	{
		m_slOut.EnableWindow(TRUE);
	}
}
```

Перейдемо модифікації обробників кнопок.

```
void CSimpleModelASUDlg::OnBnClickedBtnStart()
{
	// TODO: додайте свій код обробника повідомлень
	SetTimer(1, DELTA_TIME, NULL); // запускаємо таймер
```

```
}

void CSimpleModelASUDlg::OnBnClickedBtnPause()
{
	// TODO: додайте свій код обробника повідомлень
	KillTimer(1); // зупиняємо таймер
}

void CSimpleModelASUDlg::OnBnClickedBtnReset()
{
	// TODO: додайте свій код обробника повідомлень
	KillTimer(1);
	// Виставляємо значення за замовчуванням
	m_dH = 0.5;
	m_dInFlowRate = 0;
	m_dOutFlowRate = 0;
	m_dDrainFlowRate = 0;
	m_slIn.SetPos(0);
	m_slOut.SetPos(0);
	m_Open.SetCheck(0);
	m_Auto.SetCheck(1);
	m_slOut.EnableWindow(FALSE);
	m_dTime = 0;
}

void CSimpleModelASUDlg::OnBnClickedBtnExit()
{
	// TODO: додайте свій код обробника повідомлень
	CDialog::OnOK(); // Закриття діалогового вікна
}
```

В обробнику функції OnTimer() викликаємо функції Calc() і Draw().

```
void CSimpleModelASUDlg::OnTimer(UINT_PTR nIDEvent)
{
	// TODO: додайте свій код обробника повідомлень або виклик стандартного
	Calc();
	Draw();
	CDialogEx::OnTimer(nIDEvent);
}
```

Залишилося реалізувати дві найоб'ємніші функціїCalc() і Draw().
Почнемо з функціїCalc():

```
void CSimpleModelASUDlg::Calc()
{
	UpdateData(TRUE);
	// Розрахунок витрати рідини на виході при відкритому/закритому зливному отворі
	if (m_Open.GetCheck() == 1)
	{
		double dSpeed = sqrt(2 * 9.81 * m_dH);  // Розрахунок швидкості наповнення 
ємності рідиною
		m_dDrainFlowRate = 3.1415 * pow(0.05, 2) * dSpeed * DELTA_TIME /1000.;
	}
	else
	{
		m_dDrainFlowRate = 0;
	}
	// Надійшло в ємність m^3
	// DELTA_TIME - ms
	double Ro = 1000;
```

```
	double dInV = (m_dInFlowRate * DELTA_TIME / 1000.) / Ro;
	// Вийшло
	double dOutV = m_dDrainFlowRate * DELTA_TIME / 1000. + (m_dOutFlowRate * 
DELTA_TIME / 1000.) / Ro;
	double dV = dInV – dOutV;
	double dH = m_dH, dIntV = 0;
	double dDeltaH = 0.00001;
	double dRadius = 0.5;
	// Розрахунок рівня води у баку
	if (dV < 0)
	{
		while (fabs(dIntV) < fabs(dV))
		{
			dIntV += 3.14 * (pow(dRadius, 2) - pow(dRadius - m_dH, 2)) * 
dDeltaH;
			m_dH = m_dH - dDeltaH;
		}
	}
	else
	{
		while (fabs(dIntV) < fabs(dV))
		{
			dIntV += 3.14 * (pow(dRadius, 2) - pow(dRadius - m_dH, 2)) * 
dDeltaH;
			m_dH = m_dH + dDeltaH;
		}
	}
	// умови підтримки рівня
	if (m_dH > dRadius) m_dH = dRadius;
	if (m_dH < 0)
	{
		m_dH = 0;
		KillTimer(1); // зупиняємо моделювання, якщо ємність спорожніла
	}
	m_dTime = m_dTime + DELTA_TIME /1000.;
}
```

І останнє, що потрібно зробити, реалізувати код функціїDraw():

```
void CSimpleModelASUDlg::Draw()
{
	int nBorder = 10;   //оголошуємо змінну, що задає поля малювання
	CWnd* pWnd = GetDlgItem(IDC_DRAW);      //Виділяємо область для малювання
	CClientDC dc(pWnd);
	CRect rect;
	pWnd->GetClientRect(&rect);
	// задаємо координати області для малювання з урахуванням полів
	CRect DrawRect;
	DrawRect.left = rect.left + nBorder;
	DrawRect.top=rect.top+nBorder;
	DrawRect.right = rect.right - nBorder;
	DrawRect.bottom = rect.bottom - nBorder;

	int nWidth = rect.right – rect.left;
	int nHeight = rect.bottom – rect.top;
	// зафарбовуємо область малювання, а також рівень води у баку
	CBrush erazeBr(::GetSysColor(COLOR_MENU));
	CBrush Water(RGB(200, 200, 255));
	// створюємо об'єкт, у якому зберігається малюнок
	CBitmap Source;
	Source.CreateCompatibleBitmap(&dc, nWidth, nHeight);
```

```
	// Створюємо контекст пристрою в пам'яті
	CDC memDC;
	memDC.CreateCompatibleDC(&dc);

	memDC.SelectObject(&Source);

	memDC.SelectObject(&erazeBr);
	memDC.Rectangle(&rect);
	memDC.SelectObject(&Water);

	// малюємо контур ємності
	memDC.Arc(DrawRect,
		CPoint(nBorder + DrawRect.left, DrawRect.CenterPoint().y),
		CPoint(nBorder + DrawRect.right, DrawRect.CenterPoint().y));
	memDC.MoveTo(DrawRect.left, DrawRect.CenterPoint().y);
	memDC.LineTo(DrawRect.right, DrawRect.CenterPoint().y);

	// розраховуємо масштаб
	double dRadius = 0.5;
	double dScaleY = (DrawRect.Height() / 2.) / dRadius;
	double dScaleX = (DrawRect.Width() / 2.) / dRadius;
	int nLevel = int(dScaleY*(dRadius - m_dH));


	// малюємо лінію рівня води
	int x = int(dScaleX * sqrt(pow(dRadius, 2) - pow(dRadius - m_dH, 2)));
	memDC.MoveTo(DrawRect.CenterPoint().x - x, DrawRect.CenterPoint().y + nLevel);
	memDC.LineTo(DrawRect.CenterPoint().x + x, DrawRect.CenterPoint().y + nLevel);

	// зафарбовуємо область заповнену водою
	if (nLevel < (DrawRect.Height() / 2. - 1))
	{
		memDC.FloodFill(DrawRect.CenterPoint().x, DrawRect.bottom - 2, 0);
	}
	// Виводимо на екран значення
	CString str;
	str.Format( _T(" Рівень = % g (м)"), m_dH);
	memDC.TextOut(10, 10, str);
	str.Format(_T(" Витік = % g (кг/с)"), m_dDrainFlowRate * 1000);
	memDC.TextOut(10, 30, str);
	str.Format(_T(" Час = % g (с)"), m_dTime);
	memDC.TextOut(10, 50, str);
	// Копіює вміст контексту пристрою в пам'яті в контекст пристрою вікна
	dc.BitBlt(0, 0, nWidth, nHeight, &memDC, 0, 0, SRCCOPY);
}
```

Тепер скомпілюйте та запустіть програму. Якщо все зроблено правильно, то ви зможете робити всі дії, обумовлені в постановці завдання. Загалом робота програми виглядає, оскільки це представлено на рис 3.4.

Індивідуальне завдання

В рамках індивідуального завдання вам необхідно провести тестування роботи програми та відобразити результати у протоколі. Очевидно, що в першому варіанті вам захочеться вирішити це завдання у вигляді: запускаємо програму; натискаємо на кнопочки; бачимо як щось малюється. Все ОК. Мета даного завдання в тому, щоб ви встановили причинно-наслідковий зв'язок між дією та очікуваною реакцією.

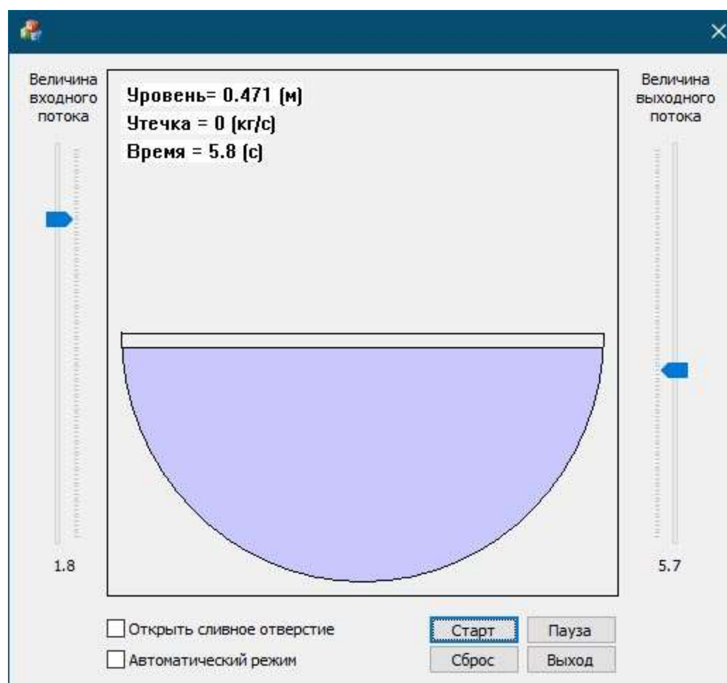


Рисунок 3.4 – Приклад роботи програми

Якщо ви розібралися у фізиці моделі процесу, то ви розумієте, до чого призведе та чи інша дія, або як ця дія може призвести до різних результатів за певних умов.

Розглянемо приклад. Дія - змінюємо витрати води на вході в ручному режимі роботи. Очікуваний результат? Якщо ви розумієте модель, то одразу можете сказати, що результат залежить від співвідношення витрати на вході та виході з урахуванням зливного отвору. Якщо ми встановлюємо витрату на вході за величиною менше сумарної витрати через труби*B* і Зто рівень води в ємності буде падати. Якщо витрата на вході більша — рівень збільшуватиметься.

Ваше завдання придумати низку перевірок, які продемонструють, що ваша програма працює правильно. Зрозуміло, що ви не можете оцінити деякі параметри абсолютної величини, але ви можете оцінити динаміку змін. Так ви не можете передбачити, з якою швидкістю буде змінюватися рівень, але ви можете передбачити, що він буде збільшуватися, зменшуватися або залишиться незмінним.

Контрольні питання

Розпишіть математичну модель процесу.

Як у моделі визначається новий рівень води у ємності?

ЛАБОРАТОРНА РОБОТА №4

Мета роботи - Вивчити механізм обміну даними (повідомленнями) між різними об'єктами програмного забезпечення.

Завдання:
1) Вивчити теоретичні відомості.
2) Скопіювати прототип програми з методичних рекомендацій
3) Реалізувати індивідуальне завдання

Теоретичні відомості

На момент виконання даної лабораторної роботи ви вже познайомилися на лекціях з поняттям об'єктно-орієнтованого аналізу та з об'єктним підходом до розробки програмного забезпечення. Кінцевою метою всього курсу лабораторних робіт є реалізувати модель процесу, розглянутого в попередній роботі з використанням об'єктно-орієнтованого підходу. Щоб не змішувати в одній роботі безліч нового матеріалу ми розглянемо в рамках окремих робіт деякі з механізмів, що використовуються в подальшому. Зокрема, у роботі ми розглянемо питання спілкування об'єктів програмного забезпечення.

Отже, якщо ми увійшли на територію об'єктно-орієнтованого підходу, то програмними об'єктами у нас, звичайно, будуть екземпляри різних класів. Для простоти, нехай це буде один класCBaseObject. Наша мета створити два екземпляри даного класу та навчити їх спілкуватися між собою.

Як же такі об'єкти можуть передавати один одному інформацію? Згадайте момент розробки інтерфейсів у попередніх роботах. Коли ви хотіли обробляти поточне положення повзунків (елементівSlider) ви створювали відповідні обробники OnVScroll() або OnHScroll(), які у свою чергу були реакцією на системні повідомлення WM_VSCROLL або WM_HSCROLL.

Кожне повідомлення, команда, подія або помилка Windows є певним кодом. Наприклад, коди повідомленьWM_VSCROLL або WM_HSCROLL відповідно 0x0115 і 0x0114. При запуску Windows генерується подія4608 - «Windows is starting up». Якщо ви намагаєтесе створити файл, який вже існує, то виникне помилка0x00B7 - «Неможливо створити файл, оскільки він уже існує».

Будь-якій вашій дії, будь-якій події у вашій програмі, будь-якій події в операційній системі відповідають певні повідомлення. Ви запустили програму, натиснули кнопку, перемістили вікно на екрані, все це генерує цілу серію різних повідомлень.

Очевидно, що разом з кожним повідомленням можуть також надсилатися різні дані. Візьмемо цей механізм на озброєння.

Спочатку введемо деякі типи даних.

1) тип даних, що перераховуєтьсяMsgCode, в якому будуть описані всі коди повідомлень, подій, комнад, станів та помилок, що надходять в об'єкт.

Фактично тип MsgCode буде безліч цілих чисел. Так приймемо, що «0» означатиме успішно виконання чогось. Значення в межах від 1 до 99 будуть ідентифікувати команди. Від 100 до 199 – інформаційні повідомлення. Від 200 до 999 – стани. Область значень від 1000 і відведемо під коди помилок.

2) тип даних, що перераховується CModelObject, в якому будуть ідентифікатори всіх об'єктів програмного забезпечення між якими необхідно реалізувати спілкування. Як і типMsgCodeце безліч цілих чисел. Для того, щоб уникнути плутанини з кодами зMsgCode почнемо нумерацію із 10000.

3) структура CMessage що складається з наступних полів:

Тип	Найменування	Призначення
MsgCode	m_nCode	код повідомлення
CModelObject	m_nObjectTo	Ідентифікатор об'єкта адресата
CModelObject	m_nObjectFrom	Ідентифікатор об'єкта відправника
void*	m_pData	Вказівник на дані

Отже, ми маємо дані для спілкування (структура CMessage) з яких випливає, що кожен об'єкт повинен мати унікальний ідентифікатор типу CModelObject. Це накладає певну вимогу до нашого класуCBaseObject. У ньому обов'язково має бути присутнім відповідний аттрибут.

Наступне питання: як надіслати повідомлення? І це найпростіше питання. Кожен об'єкт повинен мати певну функцію прийому повідомлень. Тоді надсилання повідомлення даному об'єкту означає виклик функції прийому повідомлень у даного об'єкта. Для певності назвемо цю функціюProcessMessage(). Кожне повідомлення, що надходить, необхідно спочатку проаналізувати, визначити код повідомлення, виділити і перетворити дані (якщо вони є) і т.п. Чим складніший об'єкт, тим більше кількість повідомлень, які можуть надходити в нього. Тим більше кількість різних варіантів дій і тим самим гострі потреби в попередній обробці вхідної інформації. Для такої попередньої обробки введемо якусь функціюEvents().

Розглянемо приклад. Нехай ми хочемо надіслати від об'єкту*B* об'єту *A* повідомлення "Вітання! Як ти?». Допустимо можливі варіанти відповіді: «Ок», «Так собі, але поки що тримаюся» та «Потрібна допомога»

Насамперед у нашому типі MsgCode повинен з'явитися запис з описом відповідних повідомлень:

```
enum MsgCode
{
	Success		= 0,

	// команди
	HowAreYou	= 1,

	// Повідомлення
	Middling	= 100,

	// помилки
	NeedHelp	= 1000
};
```

Очевидно, що у типі CModelObject повинні бути описані ідентифікатори наших об'єктів:

```
enum CModelObject
{
	ObjectA			= 0,
	ObjectB			= 1
};
```

Наявність у нас двох об'єктів має на увазі, що десь раніше ми створили два екземпляри класу CBaseObjectз відповідними ідентифікаторами:

```
	CBaseObject A( ObjectA );
	CBaseObject B( ObjectB );
```

Тепер створимо власне повідомлення. Якщо ми не лінуватимемося і опишемо для структуриCMessage кілька різних конструкторів, то зможемо створити потрібне повідомлення одним рядком коду:

```
CMessage msg( HowAreYou, ObjectB, ObjectA );
```

Залишилось лише надіслати це повідомлення, і для цього ми викличемо функцію прийому повідомлень у об'єкта-адресата:

```
A.ProcessMessage( msg );
```

Далі об'єкт *B*повинен запросити допомогу. Ви вже, мабуть, здогадалися, що це виглядатиме так:

```
CMessage msg( NeedHelp, ObjectA, ObjectB );
A.ProcessMessage( msg );
```

У процесі вивчення прикладу вище, у вас можливо постає питання: а як об'єкт *A* дізнається про існування об'єкта *B*і навпаки? Абсолютно правильне питання. Якщо в нашій програмі є скажімо сотня різних об'єктів, то для спілкування між собою всі вони повинні знати ілентіїкатори всіх інших об'єктів. Погодьтеся, що це не зовсім красиво, зручно і трохи коряво. Набагато зручніше створити єдиний реєстр всіх існуючих об'єктів і написати універсальну функцію, яка шукатиме в цьому реєстрі потрібного адресата та надсилатиме повідомлення саме йому. Назвемо цю функцію_SendMessage(). Зверніть увагу на те, що існує системна функціяSendMessage()і щоб їх розрізняти, ми додали символ «_» перед ім'ям функції. Звичайно, ви можете придумати для неї своє ім'я.

Як реєстр можна використовувати звичайний статичний масив покажчиків на клас CBaseObject. Як правило, максимальна кількість об'єктів у програмному забезпеченні відома. Якщо це не так, то можна використовувати і динамічний масив.

Як ви пам'ятаєте, конструктор класу повертає покажчик на створений екземпляр. Його ми і будемо використовувати для зберігання в реєстрі. Тоді створення наших об'єктів*A* і *B* буде виглядати приблизно так.

Оголошуємо глобальний реєстр-масив покажчиків:

```
#define MAXOBJECT 4
// Objects[0] - об'єкт A
// Objects[1] – об'єкт У
static CBaseObject* Objects[MAXOBJECT];
```

Тут ми використали ключове слово static для того, щоб у програмі гарантовано не було іншого об'єкта з ім'ям Objects. Таким чином використовуючи ім'яObjects ми завжди гарантовано звертатимемося саме до нашого реєстру, а не до будь-якої локальної змінної з таким самим ім'ям.

Тепер процедуру створення наших об'єктів ми можемо реалізувати так:

```
Objects[0] = new CBaseObject(ObjectA);
Objects[1] = new CBaseObject(ObjectB);
```

І оскільки масив Objects глобальний ми матимемо до нього доступ у будь-якому місці програми.

Далі стає зрозумілим алгоритм роботи функції _SendMessage(). Використовуючи ідентифікатор об'єкта одержувача повідомлення знайти у реєстрі відповідний казатель і викликати в нього методProcessMessage().

Програмно це може виглядати так:

```
for (inti = 0; i < MAXOBJECT; i++)
{
```

```
        if ((Objects[i]) && (Objects[i]->m_nIDObject == Msg.m_nObjectTo))
        {
              Objects[i]->ProcessMessage(Msg);
        }
  }
```

Тут ми перевіряємо чи існує об'єктObjects[i] і якщо так, то чи є він шуканим адресатом (Objects[i]->m_nIDObject == Msg.m_nObjectTo). І якщо обидві відповіді ствердні, то викликаємо методProcessMessage().

Звичайно, оскільки ми використовуємо динамічне виділення пам'яті під наші об'єкти, після завершення роботи програми потрібно очистити пам'ять. І тут знову ж таки немає нічого складного. Пробігаємо по нашому реєстру і при необхідності чистимо виділену раніше пам'ять:

```
for (inti = 0; i < MAXOBJECT; i++)
{
      if (Objects[i])
      {
            delete Objects[i];
      }
}
```

Отже, ми вигадали механізм спілкування програмних об'єктів, які нічого один про одного не знають. Фактично ми створюємо для кожного об'єкта набір правил на кшталт: «Якщо виникнуть проблеми, відправте Вас повідомлення "Help!". Хто такий "Вася" тобі знати не обов'язково. Повідомлення передадуть у потрібні руки наші люди».

І останнє питання з яким необхідно розібратися: як відправляти повідомлення в інтерфейс користувача? Адже на практиці це або клас діалогового вікна, або клас уявлення, причому їхні назви пов'язані з назвою проекту. Знати про існування інтерфейсу користувача нашим об'єктам також немає необхідності. Адже чим менше вони знають один про одного, тим більш універсальною стає програма в цілому. Ми можемо переписувати математику і внутрішню логіку поведінки об'єктів не дбаючи про сумісність з інтерфейсом користувача. І навпаки, можемо як завгодно змінювати інтерфейс, не дбаючи про внутрішнє наповнення програми.

Таким чином, нам необхідний якийсь проміжний об'єкт, який допоможе організувати зв'язок між конкретним класом інтерфейсу користувача і нашим механізмом спілкування об'єктів. Такий об'єкт можна реалізувати, використовуючи так званий інтерфейсний клас. Інтерфейсний клас - це клас, який не має даних і складається з віртуальних функцій. Нагадаємо, що віртуальна функція - це особливий тип функції, яка, при її виклику, виконує метод, реалізований у дочірньому класі.

У нашому випадку інтерфейсний клас виглядає дуже просто:

```
class IMsgReceive
{
public:
      virtual MsgCode ReceiveMessage(CMessage& Msg) = 0;
};
```

Ми оголошуємо одну єдину віртуальну функцію ReceiveMessage() яка є аналогом функції ProcessMessage().

Так функція віртуальна її реалізація має бути описана у класі-нащадці. Цей клас-нащадок і буде відповідати за передачу вхідних повідомлень безпосередньо інтерфейсу користувача. Свого роду це такий менеджер на ресепшені, який здає кореспонденцію від кур'єрів конкретним постояльцям. Кур'єрам не потрібно шукати конкретних постояльців, їм достатньо знати, що на вході є менеджер.

У нашому випадку клас цього менеджера може виглядати так:

```
class CMsgManager : public IMsgReceive
{
	CNameDlg* m_pDlg;
public:
	CMsgManager();
	void SetPointers(CNameDlg* pDlg);
private:
	MsgCode ReceiveMessage(CMessage& Msg);
};
```

Тут CNameDlg це назва конкретного класу інтерфейсу користувача. ФункціяSetPointers() призначена для ініціалізації вказівника m_pDlg. Коли екземпляр інтерфейсу створено, укзатель на нього необхідно помістити у змінну m_pDlg. Це дозволить отримати доступ до методів інтерфейсу. А потрібно нам це для того, що інтерфейсний клас міг викликати метод, що відповідає за обробку повідомлень, що надійшли в інтерфейс користувача. Спеціальних вимог щодо цього методу немає. Це може бути функція із довільним ім'ям. Для певності назвемо їїUpdateInterface().

Тепер, маючи екземпляр інтерфейсного класу, наша універсальна функція _SendMessage() потенційно зможе відправляти повідомлення інтерфейсу користувача.

Отже, ми розглянули механізм надсилання повідомлень між усіма об'єктами програмного забезпечення. Настав час програмної реалізації.

Програмна частина

Постановка задачі

1) Необхідно програмно реалізувати, розглянутий вище механізм взаємодії об'єктів і інтерфейсу користувача. Для певності обмежимося двома програмними об'єктами *A* і *B*.

2) Інтерфейс користувача повинен дозволяти посилати в об'єкти як мінімум два різні повідомлення.

3) Інформація про кожне повідомлення повинна відображатися на екрані.

4) Вивчити роботу механізму, дослідивши дві схеми взаємодії (див. рис. 4.1).

Схема взаємодії з варіант 1 наступна: інтерфейс користувача посилає повідомлення-команду об'єкту, а об'єкт відповідає інтерфейсу повідомленням-повідомленням про отриману команду.

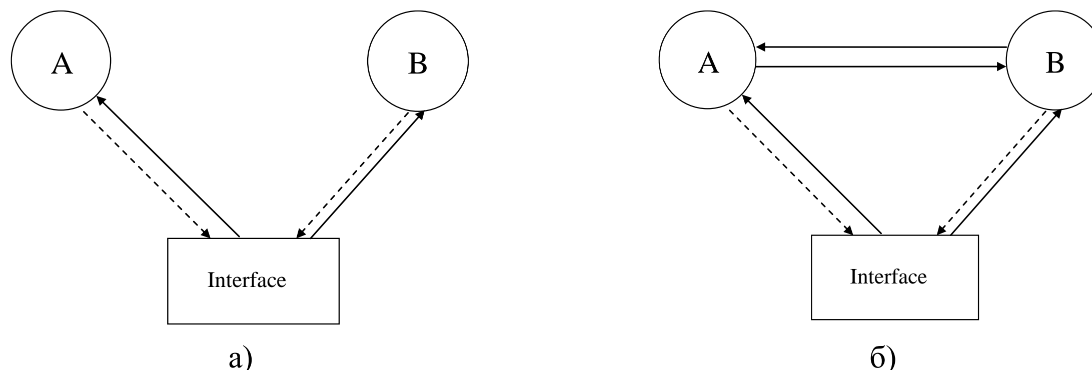


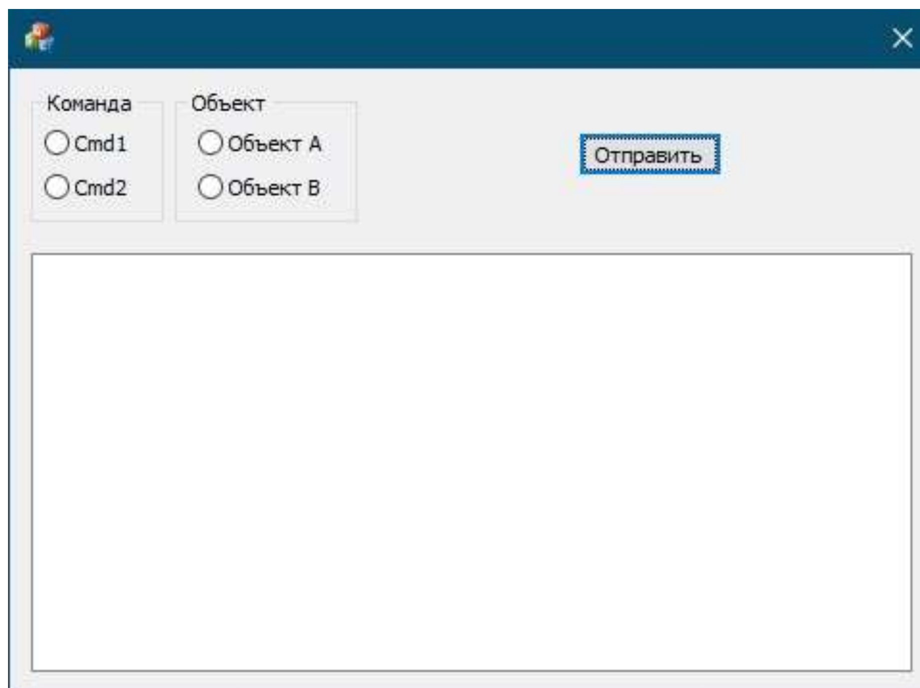
Рисунок 4.1 – Варіанти схем взаємозв'язку об'єктів: а) – варіант 1; б) – варіант 2.

Схема взаємодії варіантом 2 дещо розширена. Інтерфейс користувача посилає команду першому об'єкту. Об'єкт відповідає інтерфейсу повідомленням про отриману команду і

одночасно посилає нову команду-повідомлення другому об'єкту. Другий об'єкт відповідає інтерфейсу користувача повідомленням-повідомленням про прихід команди від першого об'єкта.

Користувальницький інтерфейс

Створіть проект HelloObjectна основі діалогового класу. Зовнішній вигляд інтерфейсу користувача представлений на рис. 4.2.



Мал. 4.2 - Вигляд інтерфейсу.

Пояснимо структуру інтерфейсу.

Для реалізації можливості вибору команд та об'єктів передбачено дві групи елементів Radio Button. Елементу"Cmd1" присвоєно ідентифікатор IDC_RB_COMMAND. Ідентифікатор елемента"Об'єкт A" - IDC_RB_OBJECT. Не забудьте в обох елементів виставити властивість«група» на значення True.

Надсилання повідомлень будемо здійснювати після натискання на кнопку IDC_BTN_SEND.

Інформацію про повідомлення, що курсують у програмі, будемо виводити в елемент List Boxз ідентифікатором IDC_PROTO. Також не зубіть вимкнути у цього елемента властивість сортування.

Далі зв'яжемо деякі елементи змінні для полегшення роботи.

Ідентифікатор	Тип	Ім'я змінної
IDC_PROTO	CListBox	m_Proto
IDC_RB_COMMAND	int	m_nCmd
IDC_RB_OBJECT	int	m_nObject

Створюємо обробник кнопки OnBnClickedBtnSend()двічі клацнувши по ній мишкою. Також нам потрібний обробникDestroyWindow() який можна додати у менеджері класів на закладці «Віртуальні функції».

Програмна реалізація

Якщо ви уважно вивчили теоретичну частину, то всі наступні дії вам не складуть труднощів. Фактично ми братимемо псевдокод, наведений раніше і адаптуватимемо його під конкретний проект.

Давайте почнемо з опису кодів всіх повідомлень, які будуть використовуватись у нашій програмі.

Додайте до проекту заголовковий файл MsgCode.h і модифікуйте його так:

```
#ifndef __MSGCODE_
#define __MSGCODE_

#pragma once

enum class MsgCode
{
	Success = 0,

	// Команди
	Cmd1 = 1,
	Cmd2 = 2,
	Cmd3 = 3,

	// Повідомлення
	NotifyAIntr = 100,  // Об'єкт A отримав команду від ПІ
	NotifyBIntr = 101,  // Об'єкт B отримав команду від ПІ
	NotifyAB = 102,     // Об'єкт A отримав команду від об'єкта
	NotifyBA = 103,     // Об'єкт B отримав команду від об'єкта A
};

#endif
```

Перше питання, яке у вас виникло це: «чому enum class, а не enum»?

Це з еволюцією мови та новим стандартом C++11. Докладніше можете почитати тут:https://ravesli.com/urok-59-klassy-enum/

Коротко пояснимо суть. Розглянемо приклад коду:

```
enum Fruits
{
	LEMON, // LEMON знаходиться всередині тієї ж області видимості, що і Fruits
	KIWI
};

enum Colors
{
	PINK, // PINK знаходиться всередині тієї ж області видимості, що і Colors
	GRAY
};
```

Формально у цьому прикладі fruit = colorпри тому, що це різні категорії, які навіть порівнювати не можна. Щоб унеможливити подібні помилки в C++11 ввеликласи enum. Тепер для використання елементів перерахування необхідно додавати перед ними ім'я класу, наприклад,`MsgCode::Cmd1`. Це значно знижує ризик конфлікту імен.

Отже, маємо набір команд. Перша і друга за нашою логікою будуть відправлятися в об'єкти з інтерфейсу користувача. Третя команда надсилатиметься об'єктами один одному.

Для того, щоб програма могла точно вказати, які події відбулися з об'єктами, ми ввели 4 відповідні повідомлення.

Очевидно, що чим складніша та детальніша система об'єктів, тим більше різних повідомлень може в ній курсувати.

Тепер додайте до проекту файл-заголовок defines.h. і модифікуйте його так:

```
#ifndef __DEFINES_
#define __DEFINES_

#pragma once
#include "pch.h"
#include "msgcode.h"

enum class CModelObject
{
	NotDefined	= 10000,
	Inteface	= 10001,
	ObjectA		= 10002,
	ObjectB		= 10003
};

struct CMessage
{
	MsgCode      m_nCode;
	CModelObject m_nObjectFrom;
	CModelObject m_nObjectTo;
	void*        m_pData;

	CMessage()
	{
		m_nCode = MsgCode::Success;
		m_nObjectFrom = CModelObject::NotDefined;
		m_nObjectTo = CModelObject::NotDefined;
		m_pData = NULL;
	};

	CMessage(MsgCode nMsg)
	{
		m_nCode = nMsg;
		m_nObjectFrom = CModelObject::NotDefined;
		m_nObjectTo = CModelObject::NotDefined;
		m_pData = NULL;
	};

	CMessage(MsgCode nMsg, CModelObject nObjectFrom, CModelObject nObjectTo)
	{
		m_nCode = nMsg;
		m_nObjectFrom = nObjectFrom;
		m_nObjectTo = nObjectTo;
		m_pData = NULL;
	};
};

#endif
```

Як зазначалося раніше, якщо ми опишемо для структури CMessageрізні конструктори, то зможемо створювати екземпляри даного типу у різний спосіб.

Якщо у вас виникло питання навіщо ми область ідентифікаторів об'єктів програми розпочали з 10000 – відповідаємо. Дивлячись на протокол роботи програми, ми зможемо легко відрізняти групи ідентифікаторів. Команди від повідомлень. Повідомлення від об'єктів тощо. Якби всі групи ідентифікаторів однаково починалися з 0, то людині було б дуже складно швидко переглядати протокол і розуміти сенс того, що відбувається без допоміжних перекладачів кодів ідентифікаторів у зрозумілі категорії.

Далі займемося реалізацією інтерфейсного класу.

Додайте до проекту файл IMsgReceive.h

```
#ifndef __IMSGRECEIVE_H
#define __IMSGRECEIVE_H

#pragma once

#include "defines.h"

struct IMsgReceive
{
	/ / Функція обробки повідомлення.
	// Параметри:
	// Msg - посилання клас повідомлення ПЗ
	// Повернення: Success – нормальне завершення, інше – код помилки.
	virtual MsgCode ReceiveMessage(CMessage& Msg) = 0;
};

#endif
```

Файли класу менеджера повідомлень можна додати двома способами. Або окремо додайте файлиMsgManager.h і MsgManager.cpp. Або додайте відразу класCMsgManager. При цьому зверніть увагу на імена файлів, які пропонуються за замовчуванням. Якщо ім'я вашого класуCMsgManager, то і назви файлів за промовчанням також будуть CMsgManager.h і CMsgManager.cpp.

Заголовний файлMsgManager.h:

```
#if !defined(_MSGMANAGER_H_)
#define _MSGMANAGER_H_

#pragma once
#include "IMsgReceive.h"

class CHelloObjectDlg;
class CMsgManager : public IMsgReceive
{
	CHelloObjectDlg* m_pDlg;
public:
	CMsgManager();
	void SetPointers(CHelloObjectDlg* pDlg);

private:
	// Функція обробки повідомлень
	// Параметри:
	// Msg – посилання клас повідомлення.
	// Повернення: Success - нормальне завершення, інше код помилки.
	virtual MsgCode ReceiveMessage(CMessage& Msg);
};
#endif
```

Файл-реалізаціяMsgManager.cpp:

```
#include "pch.h"
#include "MsgManager.h"
#include "HelloObjectDlg.h"

CMsgManager::CMsgManager()
{
	m_pDlg = NULL;
}
```

```
void CMsgManager::SetPointers(CHelloObjectDlg* pDlg)
{
	m_pDlg = pDlg;
}

MsgCode CMsgManager::ReceiveMessage(CMessage& Msg)
{
	m_pDlg->UpdateInterface(Msg);

	return MsgCode::Success;
}
```

Якщо ви захочете скомпілювати програму для перевірки на помилки, то зверніть увагу на те, що до цього моменту ще не оголошено функції.UpdateInterface(). Для успішної компіляції закоментуйте рядок із викликом.

Тепер займемося реалізацією механізму надсилання повідомлень та всього, що для цього необхідно. Це також доцільно винести на окремі файли, які ми назвемоBaseFunction.

У заголовному файлі BaseFunction.hми оголосимо функцію надсилання повідомлень, а також функцію ініціалізації інтерфейсу та його закриття. Що це за функції пояснимо трохи нижче.

```
#ifndef __BASEFUNCTION_H
#define __BASEFUNCTION_H

#pragma once

#include "defines.h"
#include "IMsgReceive.h"

MsgCode _SendMessage(CMessage& Msg);
MsgCode InitInterface(IMsgReceive* Intr);
void CloseInterface();

#endif
```

ФайлBaseFunction.cpp модифікуйте таким чином:

```
#include "pch.h"
#include "BaseFunction.h"
#include "BaseObject.h"

#define MAXOBJECT 4
// Objects[0] - об'єкт A
// Objects[1] – об'єкт У
static CBaseObject* Objects[MAXOBJECT];

// Інтерфейсний клас
static IMsgReceive* g_pInterface = NULL;

MsgCode _SendMessage(CMessage& Msg)
{
	MsgCode RetValue = MsgCode::Success;

	if (Msg.m_nObjectTo == CModelObject::Interface)
	{
		RetValue=g_pInterface->ReceiveMessage(Msg);
	}
	else
	{
```

```
		for (inti = 0; i < MAXOBJECT; i++)
		{
			if ((Objects[i]) && (Objects[i]->m_nIDObject == Msg.m_nObjectTo))
			{
				RetValue = Objects[i]->ProcessMessage(Msg);
			}
		}
	}
	return RetValue;
}

MsgCode InitInterface(IMsgReceive* Intr)		//ініціалізація моделі
{
	MsgCode RetValue = MsgCode::Success;

	for (inti = 0; i < MAXOBJECT; i++)
	{
		Objects[i] = NULL;
	}

	g_pInterface = Intr;

	Objects[0] =new CBaseObject(CModelObject::ObjectA);
	Objects[1] =new CBaseObject(CModelObject::ObjectB);

	return RetValue;
}

void CloseInterface()
{
	for (inti = 0; i < MAXOBJECT; i++)
	{
		if (Objects[i])
		{
			delete Objects[i];
		}
	}
}
```

Пояснимо окремі моменти. На початку оголошується масивObjects,який буде виконувати роль реєстру для зберігання покажчиків на об'єкти програмного забезпечення. Далі оголошується покажчик на інтерфейсний класg_pInterface.

Код функції _SendMessage() ми розглядали раніше.

У функцію ініціалізації ми передаємо як аргумент покажчик на екземпляр класу CMsgManager. Нагадаємо, що він є нащадком базового класуIMsgReceiveта цікавить нас саме як об'єкт цього базового класу. На цей момент екземпляр класуCMsgManagerвже має бути створено та ініціалізовано, тобто. його атрибутm_pDlg за допомогою методу SetPointers()повинен бути пов'язаний із діалоговим вікном. Цю ділянку коду ми розглянемо трохи згодом.

Після обнулення всіх елементів масиву Objects відбувається власне динамічне створення екземплярів наших об'єктів *A* і *B* із занесенням їх покажчиків до масиву.

Очищення пам'яті здійснюється у функції CloseInterface().

Далі перейдемо до реалізації класу наших об'єктів. Тут також ви можете додати файли окремо, так і додати відразу клас (не забудьте проконтролювати імена файлів).

Заголовний файлBaseObject.h:

```
#ifndef __BASE_OBJECT_
#define __BASE_OBJECT_
```

```
#pragma once

#include "msgcode.h"
#include "defines.h"

class CBaseObject
{
public:
	// Конструктор
	CBaseObject(CModelObject ID);
	// Деструктор
	~CBaseObject();

	CModelObject  m_nIDObject;         //ідентифікація об'єкта
	MsgCode ProcessMessage(CMessage& Msg); // функція прийому повідомлень

protected:
	//функція попередньої обробки прийнятих повідомлень
	MsgCode Events( MsgCode nCode, void* pData );
};

#endif
```

Зверніть увагу на секцію protectedв якій оголошено функцію Events(). Так як ця функція призначена тільки для аналізу того, що відбувається всередині об'єкта, ми унеможливили її виклик ззовні. Вона доступна лише всередині класуCBaseObject.

Файл-реалізація BaseObject.cpp:

```
#include "pch.h"
#include "BaseObject.h"
#include "BaseFunction.h"

// Конструктор.
CBaseObject::CBaseObject(CModelObject ID)
{
	m_nIDObject = ID;
}

// Деструктор.
CBaseObject::~CBaseObject()
{
}

/ / Функція прийому повідомлень.
MsgCode CBaseObject::ProcessMessage(CMessage& Msg)
{
	return Events(Msg.m_nCode, Msg.m_pData);
}

MsgCode CBaseObject::Events(MsgCode msg, void* pData )
{
	MsgCode nRetValue = MsgCode::Success;

	switch (msg)
	{
		case MsgCode::Cmd1:
		{
			if (m_nIDObject == CModelObject::ObjectA)
			{
				CMessage msg(MsgCode::NotifyAIntr, CModelObject::ObjectA,
CModelObject::Interface);
				_SendMessage(msg);
```

```
			}
			else if (m_nIDObject == CModelObject::ObjectB)
			{
				CMessage msg(MsgCode::NotifyBIntr, m_nIDObject,
CModelObject::Interface);
				_SendMessage(msg);
			}
		}
		break;
		case MsgCode::Cmd2:
		{
			if (m_nIDObject == CModelObject::ObjectA)
			{
				CMessage msg(MsgCode::NotifyAIntr, CModelObject::ObjectA,
CModelObject::Interface);
				_SendMessage(msg);
				CMessage msg1(MsgCode::Cmd3, CModelObject::ObjectA,
CModelObject::ObjectB);
				_SendMessage(msg1);
			}
			else if (m_nIDObject == CModelObject::ObjectB)
			{
				_SendMessage(CMessage(MsgCode::NotifyBIntrm_nIDObject,
CModelObject::Interface));
				_SendMessage(CMessage(MsgCode::Cmd3m_nIDObject,
CModelObject::ObjectA));
			}
		}
		break;
		case MsgCode::Cmd3:
		{
			if (m_nIDObject == CModelObject::ObjectA)
			{
				CMessage msg(MsgCode::NotifyBA, CModelObject::ObjectA,
CModelObject::Interface);
				_SendMessage(msg);

			}
			else if (m_nIDObject == CModelObject::ObjectB)
			{
				CMessage msg(MsgCode::NotifyAB, CModelObject::ObjectB,
CModelObject::Interface);
				_SendMessage(msg);

			}
			break;
		}
	}
	return nRetValue;
}
```

Тут особливий інтерес представляє функція Events(). Як ми вже говорили її завдання проаналізувати повідомлення, що надійшло, і прийняти виробити дію. У більш складних об'єктах ця функція може викликати безліч інших функцій, в яких реалізовані алгоритми дій для різних повідомлень, що надійшли. Але в нашому випадку алгоритм дії об'єктів тривіальний і всі дії реалізовані лише в рамках функціїEvents().

Насамперед аналізується код повідомлення, що надійшло. Це може бути три види команд (Cmd1,Cmd2,Cmd3). Алгоритм реакцію ці команди ми описали у розділі «постановка завдання».

Надалі коді показані різні варіанти реалізації алгоритму дій. Розглянь, наприклад, ділянку:

```
if (m_nIDObject == CModelObject::ObjectA)
{
	CMessage msg(MsgCode::NotifyAIntr, CModelObject::ObjectA,
CModelObject::Interface);
	_SendMessage(msg);
}
else if (m_nIDObject == CModelObject::ObjectB)
{
	CMessage msg(MsgCode::NotifyBIntr, m_nIDObject, CModelObject::Interface);
	_SendMessage(msg);
}
```

Згадайте третій конструктор структури CMessage. Другим його аргументом є ідентифікатор об'єкта, від якого надходить повідомлення. Тому визначивши за умови, що поточний об'єкт є об'єктом*A*, ми можемо використовувати відповідний ідентифікатор та перерахування CModelObject::ObjectA. З іншого боку ми можемо без жодних перевірок як другий аргумент використовувати аттрибутm_nIDObject, Який за визначенням вже має необхідне значення ідентифікатора. І якщо у першій гілці умови ми також замінимоCModelObject::ObjectA на m_nIDObject, на відміну від гілок буде тільки в коді повідомлення, що посилається. Розмірковуючи далі виникає логічне питання: а навіщо вводити різні коди повідомлень-нотифікацій, якщо ми можемо надсилати одне повідомлення, а від кого і кому воно надійшло визначати за відповідними полями структуриCMessage. У цьому випадку ми могли б скоротити наведену ділянку коду щонайменше вдвічі. Це завдання вам пропонується вирішити у рамках індивідуального завдання.

Ще один спосіб скоротити код показаний на прикладі реакції об'єкта на командуCmd2:

```
if (m_nIDObject == CModelObject::ObjectA)
{
	CMessage msg(MsgCode::NotifyAIntr, CModelObject::ObjectA,
CModelObject::Interface);
	_SendMessage(msg);
	CMessage msg1(MsgCode::Cmd3, CModelObject::ObjectA, CModelObject::ObjectB);
	_SendMessage(msg1);
}
else if (m_nIDObject == CModelObject::ObjectB)
{
	_SendMessage(CMessage(MsgCode::NotifyBIntrm_nIDObject, CModelObject::Interface));
	_SendMessage(CMessage(MsgCode::Cmd3m_nIDObject, CModelObject::ObjectA));
}
```

Нагадаємо, що при вступі цієї команди в об'єкт він повинен повідомити інтерфейс та надіслати команду Cmd3в інший об'єкт. Досі під кожне нове повідомлення ми виділяли нову змінну, і якщо у нас два різні повідомлення, то перша думка у вас швидше за все буде: «створити дві змінні», як це показано в першій гілці умови. Адже перевизначити поля структури це три рядки коду:

```
	msg.m_nCode =MsgCode::Cmd3;
	msg.m_nObjectFrom = CModelObject::ObjectA;
	msg.m_nObjectTo = CModelObject::ObjectB;
```

Але є альтернативний варіант створення екземплярів структуриCMessageбез створення змінних. Цей варіант показаний у другій гілці умови. Аргументом функції_SendMessage() є об'єкт типу CMessage. І так як у нас є аж три різні конструктори для цього типу, в даному випадку ми можемо використовувати варіант з необхідними нам аргументами:

```
_SendMessage(CMessage(MsgCode::NotifyBIntrm_nIDObject, CModelObject::Interface));
```

Це дозволяє також вдвічі скоротити обсяг коду.

Тож ми виходимо на фінішну пряму. Нам лишилося модифікувати файли класу діалогу.

Насамперед підключіть у файлі-заголовку HelloObjectDlg.h файли defines.h і MsgManager.h. Далі необхідно оголосити змінну типуCMsgManager та функцію UpdateInterface(). Зробіть це у секціїpublicвідразу після конструктора:

```
// Діалогове вікно CHelloObjectDlg
class CHelloObjectDlg : public CDialogEx
{
// Створення
public:
	CHelloObjectDlg(CWnd* pParent = nullptr);	// Стандартний конструктор

	CMsgManager m_Manager;
	void  UpdateInterface(CMessage& msg);
```

Тепер переходимо до файлу-реалізації.

Підключіть на початку файлиbaseobject.h і basefunction.h.

В обробнику OnInitDialog() проведіть ініціалізацію:

```
	// TODO: додайте додаткову ініціалізацію
	m_Manager.SetPointers(this);
	InitInterface(&m_Manager);
```

Нагадуємо, що всілякі ініціалізації в даному обробнику вносяться наприкінці після відповідного коментаря.

Далі модифікуємо обробник натискання на кнопку:

```
void CHelloObjectDlg::OnBnClickedBtnSend()
{
	// TODO: додайте свій код обробника повідомлень
	UpdateData(true);

	if (m_nObject == 0)
	{
		if (m_nCmd == 0)
		{
			_SendMessage(CMessage(MsgCode::Cmd1, CModelObject::Interface,
CModelObject::ObjectA));
		}
		else
		{
			_SendMessage(CMessage(MsgCode::Cmd2, CModelObject::Interface,
CModelObject::ObjectA));
		}
	}
	else
	{
		if (m_nCmd == 0)
		{
			_SendMessage(CMessage(MsgCode::Cmd1, CModelObject::Interface,
CModelObject::ObjectB));
		}
		else
		{
			_SendMessage(CMessage(MsgCode::Cmd2, CModelObject::Interface,
CModelObject::ObjectB));
```

```
            }
        }
}
```

Тут усе просто. Залежно від поточного значення змінних, пов'язаних із елементамиRadio Button, надсилаємо ту чи іншу команду одному з об'єктів. А для того, щоб відповідні змінні набули потрібного значення, ми спочатку ініціювали обмін даними викликавши функціюUpdateData(). Нагадуємо, що дана функція або записує в змінні значення які оператор вибрав на інтерфейсі користувача, або навпаки використовуючи поточні значення змінних вносить необхідні зміни в інтерфейс користувача.

Якщо ви подивитеся код конструктора діалогового вікна, то побачите там ініціалізацію значень змінних m_nCmd і m_nObject:

```
CHelloObjectDlg::CHelloObjectDlg(CWnd* pParent /*=nullptr*/)
        : CDialogEx(IDD_HELLOOBJECT_DIALOG, pParent)
        m_nCmd(0)
        m_nObject(0)
```

Саме завдяки цій ініціалізації під час запуску програми автоматично вибираються перші пункти в елементах Radio Button.

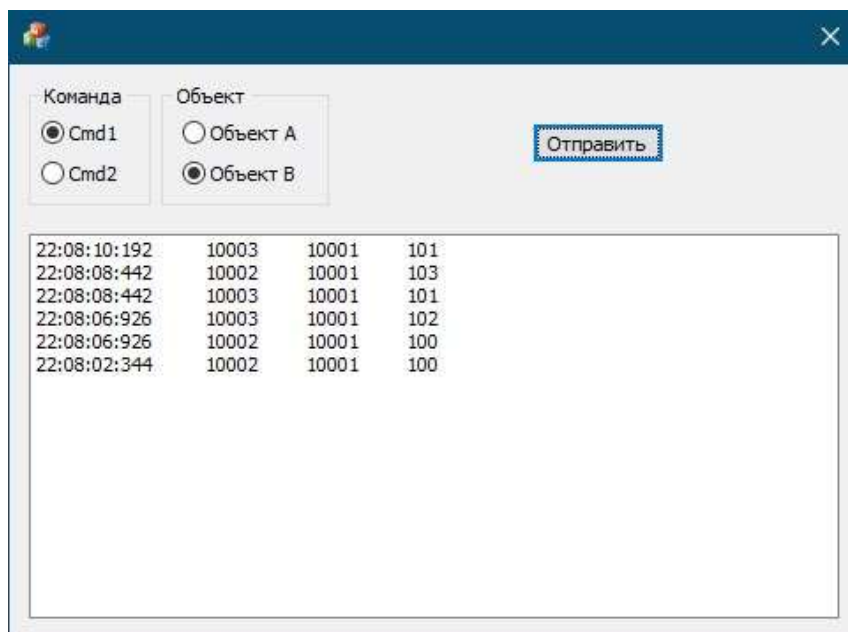
У функції UpdateInterface() ми будемо протоколювати повідомлення в елементі List Box.

```
void CHelloObjectDlg::UpdateInterface(CMessage& msg)
{
        // TODO: додайте свій код обробника повідомлень
        CString str;
        SYSTEMTIME CurTime;
        GetLocalTime(&CurTime);// Визначаємо системний час
        str.Format(_T("%2.2d:%2.2d:%2.2d:%3.3d%d%d%d"), CurTime.wHour, CurTime.wMinute,
CurTime.wSecond, CurTime.wMilliseconds, msg.m_nObjectFrom, msg.m_nObjectTo,
msg.m_nCode); // Висновок часу, код повідомлення, кому і від кого надійшло повідомлення
        m_Proto.InsertString(0, str);// Протоколуємо
}
```

І останній штрих. Необхідний обробникуDestroyWindow() викликати нашу функцію CloseOnterface(), яка очистить пам'ять, виділену при ініціалізації.

```
BOOL CHelloObjectDlg::DestroyWindow()
{
        // TODO: додайте спеціалізований код або виклик базового класу
        CloseInterface();
        return CDialogEx::DestroyWindow();
}
```

Якщо ви все зробили правильно, то результат роботи програми може виглядати так



Мал. 4.3 – Приклад результату роботи програми

Індивідуальне завдання

1) Оптимізувати код функції Events(). Для цього скоротити надлишкові повідомлення у класіMsgCode.

2) У структурі CMessageпередбачено поле як покажчика на дані. В рамках роботи це поле не використовується. Спробуйте продати варіант обміну повідомленнями з передачею деяких даних, наприклад, цілих чисел.

Контрольні питання

Поясніть механізм спілкування програмних об'єктів.

У чому відмінність в відправці повідомлень в інтерфейс користувача і програмні об'єкти?

Ви натиснули на інтерфейсі користувача кнопку «Відправити» при вибраній команді Cmd2. Простежте весь ланцюжок функцій, що викликаються починаючиз OnBnClickedBtnSend().

ЛАБОРАТОРНА РОБОТА №5

Ціль: вивчити алгоритм переходу від моделі станів до програмного коду

Завдання:
1) Вивчити теоретичні відомості
2) Реалізувати запропоновану модель станів у програмному коді
3) Виконати індивідуальне завдання

Теоретичні відомості

Модель станів відноситься до другого етапу об'єктно-орієнтованого аналізу та визначає поведінку окремо взятого об'єкта в часі. Дана модель може представлятися або у графічному, або у табличному вигляді. Графічний варіант подається у вигляді діаграми, що складається зі станів та подій (рис. 5.1). Стани відображаються овалами, події – стрілками.

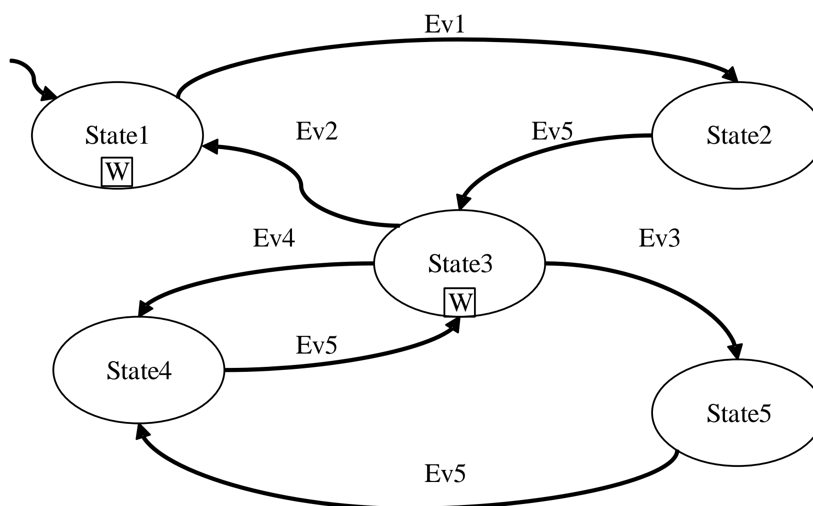
Кожна подія моделі викликає перехід об'єкта в новий стан. При досягненні нового стану об'єкт виконує певну дію. Під дією ми розуміємо той набір алгоритмів, процедур тощо. які має виконати об'єкт. У ряді випадків при переході в певні стани об'єкт нічого не робить, а чекає на нову подію. Це очікування також діє.

Всі події в моделі станів поділяються на зовнішні та внутрішні. Внутрішні події об'єкт посилає сам собі, а зовнішні приходять ззовні з інших об'єктів. Об'єкт не може обробити зовнішню подію, якщо він не перебуває у відповідному стані очікування. Уявіть, що ви граєте в гру на комп'ютері або дивиться фільм і телефонує телефон. Щоб відповісти на дзвінок вам доведеться відірватися від гри чи перегляду, тобто. перейти до нового стану, в якому ви можете відповідати на телефонні дзвінки.

Інший приклад. Уявіть, що ви готуєте на кухні щось грандіозне, щоб вразити ваших друзів. Ви кидаєтеся по всій кухні між різними станами: миєте, нарізаєте, натеряєте, ображуєте, помішуєте, додаєте спеції і т.д. І тут дзвонить телефон. І ви знаєте, що зараз не до нього. Якщо ви порушите процес, то все буде зіпсовано. Щось пригорить, щось перевариться тощо. Ви, як і раніше, переходите через різні стани, поки нарешті не перейдете в стан очікування і не зможете перевірити хто ж вам дзвонив і що йому було потрібно. Такі стани-очікування позначаються на моделі буквою "W" (рис. 5.1).

Вочевидь, що формування ліній поведінки об'єкта відповідає розробник. Саме він вирішує які потрібні стани та події і куди саме перейде об'єкт із поточного стану у разі виникнення конкретної події. Зазвичай об'єкти є відображенням деяких абстракцій реального світу та мають схожі лінії поведінки. Наприклад, об'єкт «лампочка» разом із об'єктом «кнопка» повинен спалахувати або гаснути при надходженні сигналів від кнопки. Тобто. якщо лампочка перебуває у стані «вимкнена» і надходить повідомлення «ввімкнути», необхідно перейти у стан «включена» і навпаки.

З погляду реального світу неможливо двічі включити лампочку, якщо вона вже включена. Тобто. комбінація «лампочка включена – прийшов сигнал увімкнути» неможлива. А ось із програмного погляду можливо все. Немає жодних жорстких обмежень, які забороняють двічі надіслати в лампочку повідомлення «ввімкнути». З попередньої лабораторної роботи ви пам'ятаєте, що надсилання повідомлення зводиться до виклику функціїProcessMessage()приймання повідомлення у конкретного об'єкта. Ніщо не заважає вам двічі викликати цю функцію з тим самим повідомленням. Причому якщо перше повідомлення буде коректно оброблено і призведе до переходу у потрібний стан, друге призведе до теоретично неможливої ситуації.



Малюнок 5.1. - Модель станів об'єкту

А тепер уявіть, що розробник помилився і прописав у коді невірну команду і в об'єкт "лампочка" надійшло повідомлення "обчислити синус". А раптом розробник помилився маніпулюючи пам'яттю і в результаті якогось переповнення та виходу за задані межі код повідомлення змінився на випадкове значення, що не відповідає жодному із зареєстрованих кодів. Тоді об'єкт отримає невідоме повідомлення. Як бути у всіх цих випадках?

Для виходу з подібних ситуацій модель станів вводиться стан аварії (Emergency). Перехід у цей стан можливий із будь-якого іншого. Навіть якщо явно на моделі це не позначено, воно завжди там присутній. Справа в тому, що якщо ми намалюємо на діаграмі переходи з кожного сотсояння в стан аварії, то діаграма дуже сильно потріють у своїй інформативності. Для порівняння подивіться на рис. 5.2.

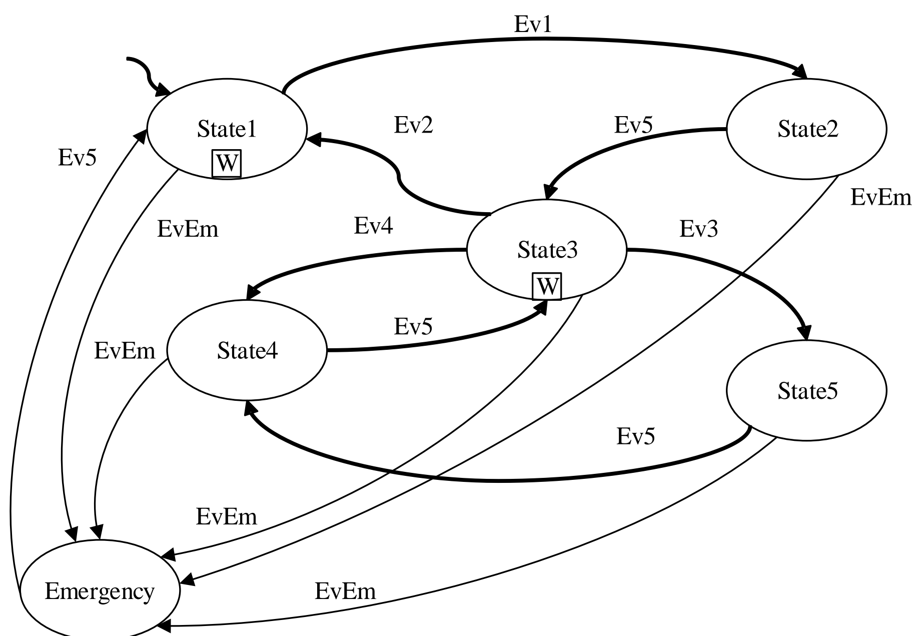


Рисунок 5.2 – Модель станів об'єкта з урахуванням стану аварії

А якщо модель описує поведінку складного об'єкта і на ній присутні десятки станів, з яких йдуть лінії в стан аварії, читати таку діаграму стане просто неможливо.

Також модель станів у вигляді діаграми поряд зі своєю наочністю і доступністю має істотний недолік. На ній можна помилково вказати неправильні переходи. З теорії ООА ви знаєте, що є певні правила формування переходів. Так, наприклад, не можна перейти з одного

стану в два інших по тому самому події. Але на діаграмі таких помилок можна не помітити. Звичайно на етапі програмування ця помилка виявиться, але на цей момент вже може бути реалізований великий шматок проекту, який, можливо, потрібно переробляти.

Щоб уникнути подібних помилок, модель станів також представляється в табличній формі. Нижче представлена таблиця переходів у стани еквівалентної моделі на рис. 5.1.

Таблиця 5.1. Таблиця переходів

	Ev1	Ev2	Ev3	Ev4	Ev5	EvEmergency
St1(W)	St2					
St2	X				St3	
St3(W)	X	St1	St5	St4		
St4	X	X			St3	
St5	X	X	X		St4	
StEmergency	X	X	X	X	St1	X

Тут необхідно зробити низку пояснень. Лівий стовпець таблиці є список всіх станів. Верхній рядок – список усіх подій. У кожному осередку позначається ідентифікатор стану, який має перейти об'єкт у разі відповідної комбінації «стан – подія». Осередки, у яких стоять ідентифікатори станів є реальні комбінації відбивають необхідне поведінка об'єкта. Решта осередків – це комбінації, які теоретично нічого не винні виникати. Але ми пам'ятаємо, що внаслідок елементарних помилок програмно ці комбінації легко відтворити.

Коли виникає будь-яка «неможлива» комбінація, у нас фактично є два шляхи. Ми можемо її проігнорувати чи перейти в аварійний стан. Який шлях вибрати у кожному разі вирішує розробник. Наприклад, можна ігнорувати команду «ввімкнути», якщо лампочка вже горить. Такі комбінації позначені у таблиці хрестиками. У всіх цих випадках ми ігноруємо події, що виникають. Але в ряді випадків розробник може вирішити, що ті чи інші комбінації свідчать про серйозні помилки у програмі. Наприклад, якщо ви керуєте ядерним реактором, то виникнення неможливих комбінацій свідчить про серйозні збої, які потенційно можуть призвести до вибуху. Тому необхідно терміново переходити в аварійний стан та зупиняти реактор.

У нашому випадку всі порожні осередки у таблиці відповідають переходам у стан аварії. Зверніть увагу також на те, що вихід зі стану аварії здійснюється за однією конкретною подією. Усі інші ігноруються. Це зроблено для того, щоб уникнути зіциклювання, коли кожне нове повідомлення має призводити до переходу в стан аварії, а в ньому вже знаходитесь.

Також таблична форма моделі станів дозволяє легко контролювати виконання різних правил та обмежень. Ви пам'ятаєте, що перехід із станів очікування здійснюється лише за зовнішніми подіями. У нашому випадку, це події Ev1-Ev4. Подія Ev5 є внутрішньою і може ініціювати переходи тільки зі станів, що не є станами очікування. Це легко візуально перевіряється у табличній формі. Тут ми фізично не можемо вказати перехід з одного стану в два інших по тому самому події. Це означало б, що в осередку повинні стояти два ідентифікатори інших станів.

Тепер, коли ми розуміємо, що таке модель станів і як вона працює, можна розглянути питання, як її перетворити на програмний код.

І так у нас є на моделі дві категорії: стан та події. І якщо стан це новий для вас момент, то з подіями ви неодноразово стикалися в попередніх лабораторних роботах, коли використовували обробники системних подій у діалогових вікнах. Вже добре знайомі вам функціїOnInitDialog(), OnButtonClicked(), OnTimer(), OnVScroll()є обробниками подій. Тому ми вчинимо так само. Для кожної події таблиці переходів ми створимо відповідні функції обробники. За аналогією та для станів необхідно зробити те саме. Ви пам'ятаєте, що з переході

у стан виконується певну дію. Ось ці дії будуть реалізовані у відповідних обробниках станів. При такому підході реалізація поведінки об'єкта зводиться у виклику в обробниках подій обробників станів і навпаки.

Алгоритм перехід від моделі станів до програмного коду

Тепер якщо зібрати разом усі знання які ви отримали у попередній лабораторній роботі та теорію розглянуту вище, ми можемо формалізувати алгоритм переходу від моделі до програмного коду.

1) Створити клас об'єкта чи ланцюжок класів з відносинами супертип-підтип. Передбачити необхідні елементи механізму обміну повідомленнями, зокрема функціїProcessMessage() і Events().

2) Оголосити елементи механізму відстеження та зміни поточного стану. У найпростішому випадку це буде якась змінна член класу об'єкта, яка прийматиме значення, що відповідають поточному стану, а також деяка функція, яка змінюватиме значення змінної.

3) Створити для кожного стану та події на моделі відповідний обробник.

4) Користуючись таблицею переходів, реалізувати переходи між станами. В обробниках станів викликатимуть відповідні обробники подій та навпаки. При цьому пам'ятайте, що стан очікування не потрібно викликати обробник події. Вихід із таких станів відбувається за зовнішньою подією.

5) Ініціювати початок життєвого циклу об'єкта шляхом виклику обробника якогось вихідного стану у конструкторі об'єкта.

Програмна частина

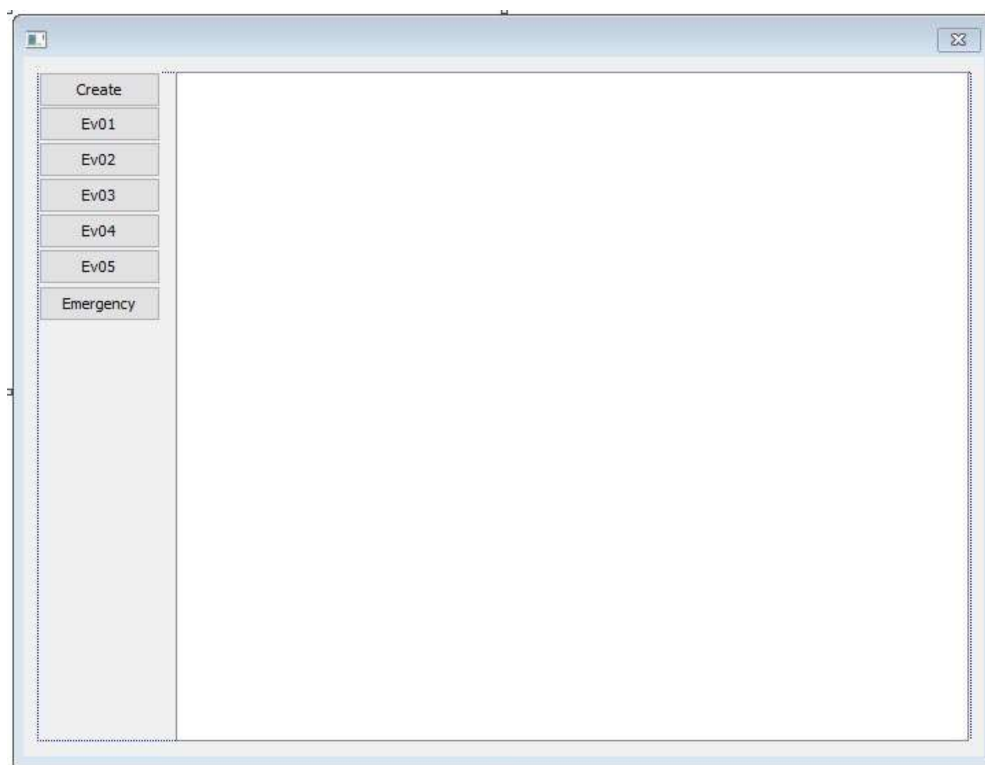
Постановка задачі

1) Необхідно розробити програмну модель об'єкта модель станів якого зображена на рис. 5.1.

2) Інтерфейс користувача повинен дозволяти вручну посилати в об'єкт будь-яке повідомлення та аналізувати його реакцію.

Користувальницький інтерфейс

На рис. 5.3 представлена форма інтерфейсу користувача, яка нам знадобиться. Зліва розташовано 7 кнопок з ідентифікаторами (зверху вниз):IDC_BTN_CREATE, IDC_BTN_EV01, IDC_BTN_EV02, IDC_BTN_EV03, IDC_BTN_EV04, IDC_BTN_EV04, IDC_BTN_EMERGENCY. Перша кнопка призначена для створення екземпляра об'єкта та ініціації його життєвого циклу. За допомогою інших кнопок ми надсилатимемо в об'єкт різні повідомлення в рамках нашої моделі.



Мал. 5.3. Вигляд інтерфейсу користувача.

Праворуч розташований елемент ListBox з ідентифікатором IDC_LST_PROTO. Не забудьте прибрати у нього якість сортування. У цей елемент виводитиметься протокол роботи програми. Для зручності роботи зв'яжіть з ним змінну m_lstProto типу CListBox.

Створити кнопки OnBtnmsg1(), OnBtnEv01(), OnBtnEv02(), OnBtnEv03(), OnBtnEv04(). Також створити обробники DestroyWindow(), OnInitDialog().

Крок 3. Переходимо до реалізації програмного коду

Для початку створимо h-файл MsgCode, в якому зберігатимемо код станів і подій, і модифікуємо його наступним чином:

```
#ifndef __MSGCODE_
#define __MSGCODE_

enum MsgCode
{
	Success =			0,

	// Події
	EvExternal1=			1,
	EvExternal2=			2,
	EvExternal3=			3,
	EvExternal4=			4,
	EvInternal5=			5,
	EvEmergency =		6,
```

```
        // Стану
        St01=                   100,
        St02=                   101,
        St03=                   102,
        St04=                   103,
        St05=                   104,
        StEmergency =           105,

        // Помилки
        ObjectNotInitiate =             1000,
        ErExecutTPS =           1001,
        ErBadData =                     1002,
};

#endif
```

Приєднуємо до об'єкта файл defines.h з лабораторного практикуму №4 наступним чином:

1) необхідно скопіювати файл до папки створеного проекту;

2) вибираємо у кладці меню $\Pr oject \to AddTo\Pr oject \to Files$. На екрані з'явиться вікно в ньому вибрати необхідний для нас файл і натиснути Ok. Аналогічно приєднуємо файл IMsgReceive.h, MsgManager.h, MsgManager.cpp.

У файлі defines.h необхідно зробити деякі зміни:

1) замість двох об'єктів A та B, оголосити один об'єкт і додати об'єкт AnyObject;

2) прибрати оголошення змінних типу CModelObject;

3) замінити останню функцію передачі повідомлення на таку:

```
CMessage( MsgCode nMsg, void * pData )  // повідомлення, яке передає код повідомлення та дані
{
        m_nCode = nMsg;
        m_pData = pData;
};
```

4) додати структуру CErrorData для знаходження помилок:

```
struct CErrorData
{
        int     nLine;
        char* pText;
};
```

У трьох інших файлах необхідно в програмному коді змінити назву проекту (CLab4Dlg на CSimpleOOModelDlg). Підключити в конструкторі SimpleOOModelDlg.h змінну класу CMsgManager m_Manager.

Створюємо h-файл baseobject, який є базовим класом, і модифікуємо його наступним чином:

```
#ifndef __BASEOBJECT_H
#define __BASEOBJECT_H

#include "msgcode.h"
#include "defines.h"
#include "IMsgReceive.h"

class CBaseObject
{
protected:
	bool		m_bInit;	// Ознака успішної ініціалізації об'єкта
	MsgCode			m_nState;		// Стан об'єкта.
	IMsgReceive*	m_pIntr;

	MsgCode ChangeState(MsgCode nState);
	void ProtoMessage (CMessage & Msg); //функція протоколювання
події

	MsgCode Events (MsgCode nCode, void * pData); // Функція
попередньої обробки прийнятих повідомлень
private:

	// обробники подій
	MsgCode Ev01();
	MsgCode Ev02();
	MsgCode Ev03();
	MsgCode Ev04();
	MsgCode Ev05();
	MsgCode EvEmergency( MsgCode nCode, int nNumLine, char*
lpFileName );

	// обробники станів
	MsgCode ActionSt01();
	MsgCode ActionSt02();
	MsgCode ActionSt03();
	MsgCode ActionSt04();
	MsgCode ActionSt05();
```

MsgCode ActionEmergency(MsgCode nCode, int nNumLine, char* lpFileName);

```
public:
        virtual MsgCode ProcessMessage(CMessage& Msg);// функція для отримання зовнішніх повідомлень
        // конструктор
        CBaseObject(IMsgReceive * pIntr);
        // Деструктор
        ~CBaseObject();
};

#endif
```

Створюємо cpp-файл baseobject, який є базовим класом, і модифікуємо його так:

```
#include "baseobject.h"
#include <stdio.h>

void CBaseObject::ProtoMessage( CMessage& Msg )
{
        m_pIntr->ReceiveMessage(Msg);
}

// Функція прийому повідомлень.
// Параметри:
// Msg – посилання повідомлення.
//Повернення:
// Success – нормальне завершення, інше – код завершення.
MsgCode CBaseObject::ProcessMessage( CMessage& Msg )
{
        // Перевіряємо успішність ініціалізації.
        if (!m_bInit) return ObjectNotInitiate;
        ProtoMessage(Msg);

        return Events( Msg.m_nCode, Msg.m_pData );
}

// Змінити стан.
// Параметри:
/ / nState - новий стан.
//Повернення:
// Success – нормальне завершення, інше – код завершення.
MsgCode CBaseObject::ChangeState(MsgCode nState)
{
```

```
        // Змінюємо стан.
        m_nState = nState;
        CMessage msg(nState);
        ProtoMessage(msg);

        return Success;
}

// Контруктор.
CBaseObject::CBaseObject( IMsgReceive* pIntr )
{
        m_pIntr = pIntr;
        ActionSt01();
}

// Деструктор.
CBaseObject::~CBaseObject()
{
}

MsgCode CBaseObject::Events( MsgCode msg, void* pData )
{
        MsgCode nRetValue = Success;

        switch (msg)
        {
        case EvExternal1:
                {
                        nRetValue = Ev01();
                }
                break;
        case EvExternal2:
                {
                        nRetValue = Ev02();
                }
                break;
        case EvExternal3:
                {
                        nRetValue = Ev03();
                }
                break;
        case EvExternal4:
                {
                        nRetValue = Ev04();
                }
```

```
                break;
                default:
                {
                        nRetValue = EvEmergency( ErBadData, __LINE__,
__FILE__ );
                }
                break;
        }
        return nRetValue;
    }
    // обробники подій заповнюються відповідно до моделі станів
    MsgCode CBaseObject::Ev01()
    {
        MsgCode RetValue = Success;
        switch (m_nState)
        {

        case St01:
                {
                        RetValue = ActionSt02();
                }
                break;
        default:
                {
                        // Ігнорується
                }
                break;
        }
        return RetValue;
    }

    MsgCode CBaseObject::Ev02()
    {
        MsgCode RetValue = Success;
        switch (m_nState)
        {
        case St01:
        case St02:
                {
                        RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
                }
                break;
        case St03:
                {
```

```
                        RetValue = ActionSt01();
                }
                break;

        default:
                {
                        // Ігнорується
                }
                break;
        }
        return RetValue;
}

MsgCode CBaseObject::Ev03()
{
        MsgCode RetValue = Success;
        switch (m_nState)
        {
        case St01:
        case St02:
        case St04:
                {
                        RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
                }
                break;
        case St03:
                {
                        RetValue = ActionSt05();
                }
                break;
                default:
                {
                        // Ігнорується
                }
                break;

        }
        return RetValue;
}

MsgCode CBaseObject::Ev04()
{
        MsgCode RetValue = Success;
        switch (m_nState)
```

```
			{
			case St01:
			case St02:
			case St04:
			case St05:

				{
					RetValue = ActionEmergency( ErExecutTPS, __LINE__, 
__FILE__ );
				}
				break;
			case St03:
				{
					RetValue = ActionSt04();
				}
				break;
				default:
				{
					// Ігнорується
				}
				break;
			}
			return RetValue;
		}

		MsgCode CBaseObject::Ev05()
		{
			MsgCode RetValue = Success;
			switch (m_nState)
			{
			case St01:
			case St03:

				{
					RetValue = ActionEmergency( ErExecutTPS, __LINE__, 
__FILE__ );
				}
				break;
			case St02:
			case St04:
				{
					RetValue = ActionSt03();
				}
				break;
			case St05:
```

```
				{
					RetValue = ActionSt04();
				}
				break;
			case StEmergency:
				{
					RetValue = ActionSt01();
				}
				break;
				default:
				{
					// Ігнорується
				}
				break;
			}
			return RetValue;
		}

		MsgCode CBaseObject::EvEmergency( MsgCode nCode, int nNumLine, char* lpFileName )
		{
			MsgCode RetValue = Success;
			switch (m_nState)
			{
			case StEmergency:
				{
					// Ігнорується
				}
				break;
			default:
				{
					RetValue = ActionEmergency( nCode, nNumLine, lpFileName );
				}
				break;
			}

			return RetValue;
		}

		// обробники станів заповнюються відповідно до моделі станів
		MsgCode CBaseObject::ActionSt01()
		{
			ChangeState( St01 );
			return Success;
```

```
}

MsgCode CBaseObject::ActionSt02()
{
	ChangeState( St02 );
	return Ev05();
}

MsgCode CBaseObject::ActionSt03()
{
	ChangeState( St03 );
	return Success;
}

MsgCode CBaseObject::ActionSt04()
{
	ChangeState( St04 );
	return Ev05();
}

MsgCode CBaseObject::ActionSt05()
{
	ChangeState( St05 );
	return Ev05();
}

MsgCode CBaseObject::ActionEmergency( MsgCode nCode, int nNumLine,
char* lpFileName )
{
	ChangeState( StEmergency );

	CErrorData erData;
	erData.nLine = nNumLine;
	erData.pText = lpFileName;

	CMessage msg( nCode, &erData );
	ProtoMessage(msg);

	return Ev05();
}
```

Необхідно оголосити посилання на змінну класу CBaseObject у конструкторі файлу SimpleOOModelDlg.h та підключити до нього BaseObject.h та MsgManager.h.

CBaseObject *m_pObject;

Приступаємо до завершального етапу нашого проекту: працюємо з файлом SimpleOOModelDlg.

– проводимо початкову ініціалізацію нашої моделі у конструкторі діалогового вікна:

```
m_pObject = NULL;
```

– модифікуємо OnInitDialog():

```
m_Manager.SetPointers(this);
```

– модифікуємо обробник DestroyWindow():

```
if (m_pObject) delete m_pObject;       // видаляємо об'єкт
```

Переходимо до обробників кнопок. Модифікуємо їх так:

```
void CSimpleOOModelDlg::OnBtnmsg1()
{
        // TODO: Add your control notification handler code here
        if ( !m_pObject )  // Створюємо новий об'єкт і входимо в модель станів
        {
                m_pObject = new CBaseObject(&m_Manager);
        }
}
void CSimpleOOModelDlg::OnBtnEv01()
{
        // TODO: Add your control notification handler code here
        CMessage msg(EvExternal1);  // створюємо повідомлення з кодом події
        m_pObject->ProcessMessage(msg);   // Передаємо в модель станів
}
void CSimpleOOModelDlg::OnBtnEv02()
{
        // TODO: Add your control notification handler code here
        CMessage msg(EvExternal2);
        m_pObject->ProcessMessage(msg);
}
void CSimpleOOModelDlg::OnBtnEv03()
{
        // TODO: Add your control notification handler code here
        CMessage msg(EvExternal3);
        m_pObject->ProcessMessage(msg);
}
void CSimpleOOModelDlg::OnBtnEv04()
{
```

```
		// TODO: Add your control notification handler code here
		CMessage msg(EvExternal4);
		m_pObject->ProcessMessage(msg);
	}
```

Оголошуємо функцію UpdateInterface у h-файлі діалогу та реалізовуємо її у cpp-файлі:

```
	char szBuf[256];
	SYSTEMTIME CurTime;
	GetLocalTime(&CurTime);
	switch ( msg.m_nCode )			// висновок у ListBox повідомлення
	{
		case St01:
		case St02:
		case St03:
		case St04:
		case St05:
		case StEmergency:
			{
				sprintf( szBuf, "%2.2d/%2.2d/%d%2.2d:%2.2d:%2.2d:%3.3d Стан змінився на: %d", CurTime.wDay, CurTime.wMonth, CurTime.wYear, CurTime.wHour, CurTime.wMinute, CurTime.wSecond, CurTime.wMilliseconds, msg.m_nCode );
				m_lstProto.AddString(szBuf);
			}
			break;
		case EvExternal1:
		case EvExternal2:
		case EvExternal3:
		case EvExternal4:
		case EvInternal5:
			{
				sprintf( szBuf, "%2.2d/%2.2d/%d%2.2d:%2.2d:%2.2d:%3.3d Надійшло повідомлення: %d", CurTime.wDay, CurTime.wMonth, CurTime.wYear, CurTime.wHour, CurTime.wMinute, CurTime.wSecond, CurTime.wMilliseconds, msg.m_nCode);
				m_lstProto.AddString(szBuf);
			}
		break;
		default:
			{
				sprintf( szBuf, "%2.2d/%2.2d/%d%2.2d:%2.2d:%2.2d:%3.3d Невідоме повідомлення:", CurTime.wDay, CurTime.wMonth, CurTime.wYear, CurTime.wHour, CurTime.wMinute, CurTime.wSecond, CurTime.wMilliseconds);
```

```
            m_lstProto.AddString(szBuf);
        }
}
```

ЛАБОРАТОРНА РОБОТА №6

У цьому лабораторному практикумі ми розглянемо третю Лабораторну роботу з погляду объектно-ориентированного аналізу. Для цього ознайомимося з поняттями інформаційна модель та раніше вивченою моделлю станів.

Дано: модель стенду з вхідним та вихідним отвором та зливний отвір, наповнена рідиною.

Необхідно:

- Розробити об'єктно-орієнтовану модель стенду;
- реалізувати її в програмний код;
- Провести тестування.

Завдання лабораторного практикуму:

Крок 1. Створити проект.

Крок 2. Розробка інтерфейсу користувача

Крок 3. Реалізація програмного коду

Теоретичні відомості

I. Визначення

З теорії вам відомо, що об'єктно-орієнтоване проектування включає розробку:

- Інформаційної моделі;
- моделі станів;
- алгоритмів дій.

p align="justify"> Інформаційна модель повинна показує, з яких об'єктів складається ГО модель стенду, як ці об'єкти пов'язані один з одним, які атрибути цих об'єктів.

Модель станів має на увазі наявність двох складових: діаграми станів та таблиці станів. Моделі станів складаються кожного об'єкта інформаційної моделі.

Основне завдання моделі станів - показати в яких станах може бути об'єкт, які переходи між станами можливі, а також які події викликають ці переходи.

Нагадаємо, що у кожному стані об'єкти виконують певні дії. Дії можуть бути простими та очевидними такими як: чекати, зрадити, отримати тощо. Але деякі дії, наприклад, розрахунок деякої величини, можуть бути складними та неочевидними. Нагадаємо, що мета ГО проектування — дати програмісту вичерпну інформацію, достатню для реалізації моделі в програмному коді. Для цього проектувальник, у разі потреби, наводить необхідні алгоритми дій.

У нашому випадку таким алгоритмом слід забезпечити стан, у якому здійснюватиметься розрахунок.

II. Концепція

Насамперед, ми повинні визначити глобальні механізми та структури даних, які ляжуть в основу ПЗ.

Скажімо, модель нашого стенду складається з одного об'єкта – об'єкта управління (ОУ). ОУ відповідатиме за моделювання технологічного процесу розглянутого у третьому лабораторному практикумі.

Від інтерфейсу ОУ будуть приходити команди, які надходитимуть у вигляді повідомлень (Передача повідомлень докладно розглянута в лабораторному практикумі №4).

Введемо можливість протоколювання та познайомимося з поняттям датчиків.

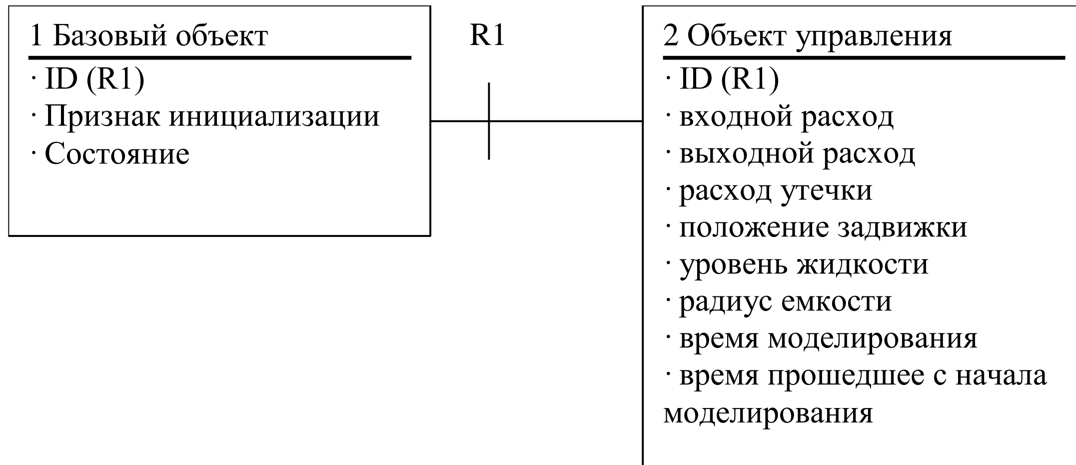
III. Об'єктно-орієнтована модель стенду

3.1. Інформаційна модель

Визначимося з атрибутами ОУ:

- Витрата рідини на вході ємності;
- Витрата рідини на виході ємності;
- Витрата рідини через зливний отвір;
- Положення засувки на зливному отворі;
- Рівень рідини в ємності;
- Радіус ємності;
- Час моделювання однієї ітерації;
- Час, що минув з початку моделювання.

Отже, наша інформаційна модель набуде вигляду (рис. 6.1.).

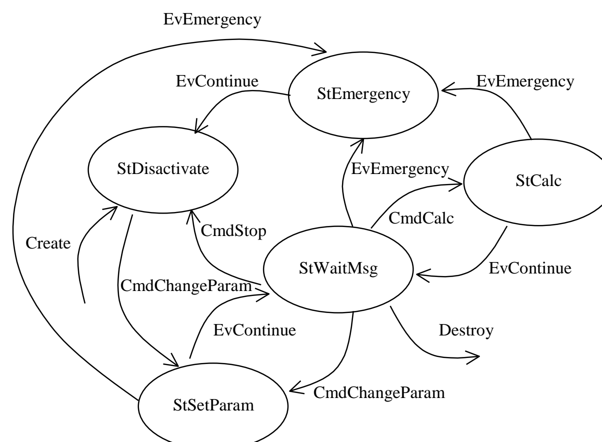


Малюнок 6.1. Інформаційна модель

3.2. Модель станів

Потрібно уявити собі життєвий цикл ОУ. Коли буде створюватися екземпляр об'єкта ОУ, він автоматично переходитиме в якийсь неактивний стан, що сигналізує про те, що об'єкт вже створений і готовий до початку роботи. У цьому стані ОУ чекатиме приходу команди активації. Ця команда ініціюватиме перехід в основний стан очікування команд. У цьому стані ОУ чекатиме приходу команд на перерахунок параметрів моделі та зміну цих параметрів. Необхідно також врахувати стан аварії, яка може запобігти помилкам.

Модель станів та таблиця переходів представлені малюнку 6.2. а,б. Опис складання моделі станів та таблиці переходів докладно було розглянуто у лабораторному практикумі №5.



		EvContinue	CmdStop	CmdCalc	CmdChangePara	EvEmergency
1	StDisactivate (W)	X	X	X	3	X
2	StWaiteMsg (W)		1	4	3	
3	StSetParam	2				
4	StCalc	2				
5	StEmergency	1	X	X	X	X

Малюнок 6.2. Модель станів та таблиця переходів

3.3. Алгоритм розрахунку представлений у лабораторному практикумі №5

IV. Специфікація функцій

Специфікації функцій зазвичай розробляються кожного об'єкта інформаційної моделі. Згадайте рис. 4. У нас є два об'єкти. Супертип «Базовий об'єкт» та підтип «Об'єкт управління». Відповідні класи назвемо CBaseObject і CControlObject.

Клас CBaseObject крім атрибутів вказаних в інформаційній моделі містите наступний ряд функцій:

Прототип функції	Опис
virtual MsgCode Events(MsgCode nCode, void * pData)	Функція обробки подій, що перевизначається. Параметри: nCode – код події. pData – покажчик на дані. Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ChangeState(MsgCode nState);	Змінити стан. Параметри: nState - новий стан. Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.

virtual MsgCode ProcessMessage(CMessage& Msg);	Функція отримання повідомлень Параметри: Msg – повідомлення. Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.

Слово virtual у прототипах функцій означає, що ця функція віртуальна. Тобто ми говоримо, що така функція в принципі буде, але що саме виконує ця функція, визначається в класі-спадкоємці. У нашому випадку клас CControlObject.

Слід зазначити, що розробити повний перелік всіх функцій у цій стадії практично дуже складно. У ході програмної реалізації завжди виникають певні уточнення та ідеї, які можуть призводити до необхідності запровадження додаткових функцій. Які своєю чергою або розширюють можливості ПЗ, або спрощують реалізацію, або роблять програмний код простим читання.

Специфікація для класу CControlObject буде значно ширшою. Основу цієї специфікації насамперед складуть дані моделі станів. Кожній події та стану на моделі відповідає функція-обробник. Для простоти і зручності ми дамо цим обробникам ті ж імена, що використані в моделі станів. Тоді специфікація набуде наступного вигляду:

Прототип функції	Опис
MsgCode EvContinue()	Обробник події «EvContinue» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode EvStop()	Обробник події «EvStop» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode EvStart()	Обробник події «EvStart» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode EvCalc()	Обробник події «EvCalc» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.

MsgCode EvChangeParam(MsgCode nCode, void* pData)	Обробник події «EvChangeParam» Параметри: nCode - код повідомлення pData - покажчик на дані Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode EvEmergency(MsgCode nCode, int nNumLine, char* lpFileName)	Обробник події «EvEmergency» Параметри: nCode - код повідомлення nNumLine – номер рядка, в якому сталася помилка lpFileName – ім'я файлу, в якому сталася помилка Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ActionDisactivate()	Обробник стану «ActionDisactivate» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ActionWaitMsg()	Обробник стану «ActionWaitMsg» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ActionSetParam(MsgCode nCode, void * pData)	Обробник стану «ActionSetParam» Параметри: nCode - код повідомлення pData - покажчик на дані Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ActionCalc()	Обробник стану «ActionCalc» Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.
MsgCode ActionEmergency(MsgCode nCode, int nNumLine, char* lpFileName)	Обробник стану «ActionEmergency» nNumLine – номер рядка, в якому сталася помилка lpFileName – ім'я файлу, в якому сталася помилка Повернення: 0 - нормальне завершення, інше - код помилки, що відбулася.

Крім того, нам знадобляться функції формату double GetRadius.

Далі, з концепції, нам знадобиться інтерфейсний клас. Назвемо його IMsgReceive. Цей клас міститиме одну віртуальну функцію. Ось її прототип:

virtual MsgCode ReceiveMessage(CMessage& Msg)	Функція прийому повідомлень від ОУ та СУ Параметри: Msg – повідомлення

Цей клас буде базовим. Його властивості будуть успадковувати інший клас, який ми назвемо CMsgManager. Функцію ReceiveMessage() буде перевизначено в класі CMsgManager. Саме в ньому буде встановлений зв'язок між даними повідомлень і конкретними елементами управління інтерфейсу користувача.

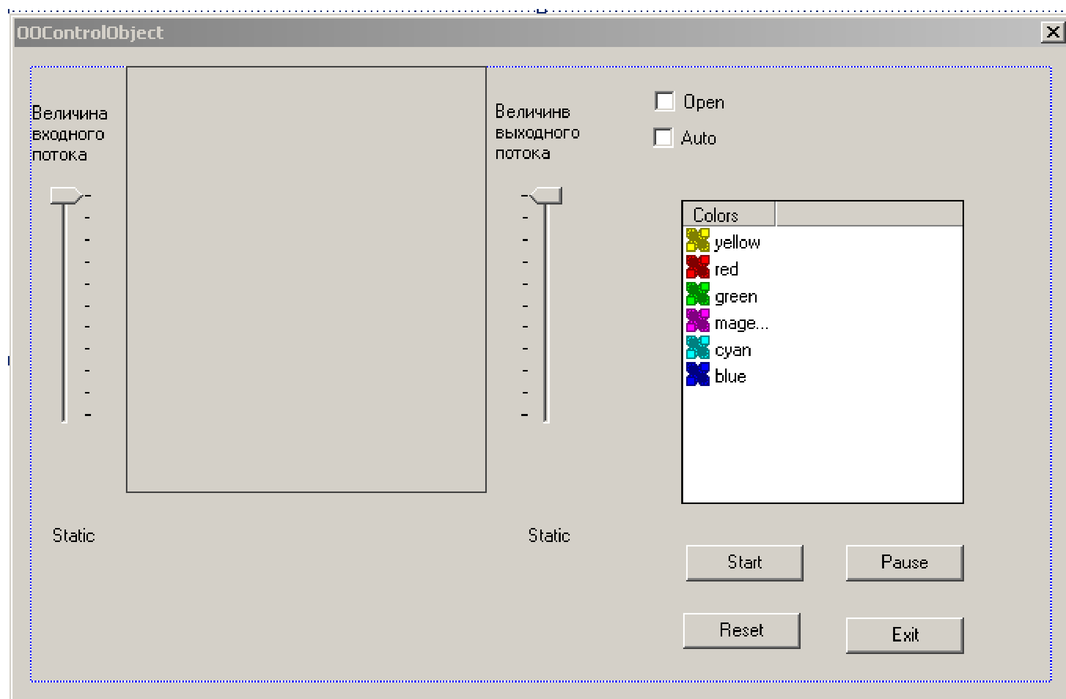
Програмна частина

V. Реалізація програмного коду

Крок 1. Створити проект OOControlObject.

Крок 2. Розробка інтерфейсу користувача

!Нагадуємо. Щоб інтерфейс підтримував російську мову, необхідно перед змінами в діалоговому вікні вказати у властивостях вікна російську мову.



Малюнок 6.3. - Користувальницький інтерфейс

На рис. 3.1 показаний зовнішній вигляд інтерфейсу, на якому представлені такі елементи:

- Повзунки (SliderBar): IDC_SLIDER_IN, IDC_SLIDER_OUT;
- кнопки (Button): IDC_BTN_START, IDC_BTN_PAUSE, IDC_BTN_RESET, IDC_BTN_EXIT;
- аркуш (ListControl): IDC_LIST;
- елементи (CheckBox): IDC_CHECK_OPEN, IDC_CHECK_AUTO

Полям Static text за умовчанням надається ідентифікатор IDC_STATIC, і навіть IDC_IN_VALUE, IDC_OUT_VALUE. У властивостях двох останніх вказати Group. Поля для малювання (Picture) надається значення IDC_DRAW.

Надаємо ідентифікаторам ім'я змінної

Ідентифікатор	**Тип**	**Ім'я змінної**
IDC_CHECK_AUTO	CButton	m_Auto
IDC_CHECK_OPEN	CButton	m_Open
IDC_SLIDER_IN	CSliderCtrl	m_slIn
IDC_SLIDER_OUT	CSliderCtrl	m_slOut
IDC_LIST	CListCtrl	m_SensorList

Створюємо обробники кнопок OnBtnStart(), OnBtnPause(), OnBtnReset(), OnBtnExit(), OnCheckAuto(), OnCheckOpen() двічі клацнувши на кнопці мишкою. Також створити обробники OnInitDialog(), OnTimer(), OnVScroll() таким чином: $View \Rightarrow ClassWizard \Rightarrow MessageMaps$. Вибираємо у Messages WM_INITDIALOG, натискаємо AddFunction. Такі самі операції виконуємо для двох інших функцій.

Крок 3. Реалізація програмного коду

Створюємо h-файл (MsgCode), у якому зберігатимуться коди повідомлень, і модифікуємо його так:

```
#ifndef __MSGCODE_
#define __MSGCODE_

enum MsgCode
{
	Success =			0,

	// Команди
	CmdStop =			1,
	CmdCalc =			2,
	CmdChangeInFlowRate =	3,
	CmdChangeOutFlowRate =	4,
```

```
		CmdChangeH =			5,
		CmdChangeDrainSwitch =	6,
		CmdChangeTime =			7,
		CmdStart =				8,
		CmdUpdate =				9,
		CmdRadius =				10,

		//Стану
		StDisactivate =			100,
		StWaitMsg =				101,
		StSetParam =			102,
		StCalc =				103,
		StEmergency =			104,

		// Помилки
		ObjectNotInitiate =		1000,
		ErExecutTPS =			1001,
		ErBadData =				1002,
		ErCalc =				1003,
	};

	#endif
```

Створюємо h-файл (defines), в якому буде описано структуру передачі повідомлень CMessage, модифікуємо його наступним чином:

```
	#ifndef __MESSAGE_INFO_
	#define __MESSAGE_INFO_

	#include "MsgCode.h"
	#include "stdafx.h"

	enum CModelObject
	{
		// Об'єкти ПЗ
		NotDefined =			10000,
		Interface =				10001,
		ControlObject =			10002,
	};

	struct CMessage
	{
		MsgCode			m_nCode;
		CModelObject	m_nObjectFrom;
		CModelObject	m_nObjectTo;
		int				m_nSize;
```

```
        void*                  m_pData;


        CMessage( MsgCode nMsg, CModelObject nObjectTo, CModelObject
nObjectFrom )
        {                   // Спеціальна функція, що передає код
повідомлення
            m_nObjectFrom = nObjectFrom;
            m_nObjectTo=nObjectTo;
            m_nCode = nMsg;
        };
        CMessage( MsgCode nMsg, CModelObject nObjectTo, CModelObject
nObjectFrom, void* pData, int nSize )
        {                   // спеціальна функція, що передає код
повідомлення від кого та кому, дані та розмір

            m_nObjectFrom = nObjectFrom;
            m_nObjectTo=nObjectTo;
            m_nCode = nMsg;
            m_pData = pData;
            m_nSize = nSize;
        };
    };

    #endif
```

Приєднуємо файли до проекту попереднього лабораторного практикуму MsgManager.h, MsgManager.cpp, IMsgReceive.h.

Створюємо h-файл SensorData, в якому зберігатимуться датчики об'єкта управління, та модифікуємо його наступним чином:

```
    #ifndef __SENSOR_DATA_H_
    #define __SENSOR_DATA_H_

    struct CControlObjectSensors
    {
        double          InFlowRate;         // витрати води на вході, кг/с
        double          OutFlowRate;            // Витрата води на виході,
кг/с
        double          DrainFlowRate;      // Витрата води на сливі, кг/с
        BOOL            DrainSwitch;            // стан зливного отвору
(відкрито 1)
        double          H;                  // рівень води у резервуарі, м
        double          TimeFromBegining;       // Час моделювання, з
    };
```

```
#endif
```

Приєднуємо h-файл BaseObject із лабораторного практикуму №4. В атрибутах доступу базового класу вносимо деякі додатки:

```
– protected:
bool            m_bInit;     // Ознака успішної ініціалізації об'єкта
MsgCode         m_nState;    // Стан об'єкта.
virtual MsgCode Events( MsgCode nCode, void * pData) = 0;
// Змінити стан.
// Параметри:
/ / nState - новий стан.
// Повернення:
// Success - нормальне завершення, інше-код завершення.
MsgCode ChangeState(MsgCode nState);
- public:
/ / Оператор порівняння класу з ідентифікатором об'єкта.
bool operator==(CModelObject nIDObject )
{
        return (m_nIDObject == nIDObject);
}
```

Приєднуємо файл basefunction.h з лабораторного практикуму №4 та вносимо деякі доповнення:

- Підключаємо файл SensorData.h;

– у функцію InitModel() додаємо покажчик pSensorsData на блок датчиків об'єкта управління:

```
MsgCode InitModel( IMsgReceive *Intr, CControlObjectSensors *pSensorsData );
```

Приєднуємо файл BaseObject.cpp із лабораторного практикуму №4 та вносимо деякі доповнення:

– у функцію ProcessMessage перевіряємо успішність ініціалізації:

```
if (!m_bInit) return ObjectNotInitiate;
```

- Додаємо функцію, яка змінює стан об'єкта:

```
// Змінити стан.
// Параметри:
/ / nState - новий стан.
// Повернення:
// Success – нормальне завершення, інше – код завершення.
```

```
MsgCode CBaseObject::ChangeState(MsgCode nState)
{
        // Стан змінилося.
        m_nState = nState;

        return Success;
}
```

– у конструкторі базового класу додаємо початкову ініціалізацію: m_bInit = TRUE;

- Видаляємо обробник функції Events () (Він буде реалізований в класі CControlObject).

Створюємо cpp-файл basefunction і модифікуємо його наступним чином:

```
#include "BaseFunction.h"
#include "IMsgReceive.h"
#include "SensorData.h"
#include "ControlObject.h"

#define MAXOBJECT 4
// Objects[0] - об'єкт управління
// Objects[1] - система управління
static CBaseObject *Objects [MAXOBJECT];

// Інтерфейсний клас
static IMsgReceive * g_pInterface = NULL;

CControlObjectSensors *g_pSensorsData = NULL;

// Надіслати повідомлення об'єкту ПЗ.
// Msg – нове повідомлення
// Повернення: Success - нормальне завершення, інше – код завершення.
// функція шукає у списку об'єкт в потрібний і передає йому повідомлення.
MsgCode _SendMessage( CMessage& Msg )
{
        MsgCode RetValue = Success;

        if ( Msg.m_nObjectTo == Interface )
        {
                RetValue = g_pInterface->ReceiveMessage( Msg );
        }
        else
        {
                for (int i = 0; i < MAXOBJECT; i++ )
```

```
                {
                        if (( Objects[i] ) && ( Objects[i]->m_nIDObject ==
Msg.m_nObjectTo ))
                        {
                                RetValue = Objects [i] -> ProcessMessage (Msg);
                        }
                }
        }
        return RetValue;
}

MsgCode InitModel( IMsgReceive *Intr, CControlObjectSensors
*pSensorsData )
{
        MsgCode RetValue = Success;


        for (int i = 0; i < MAXOBJECT; i++ )
        {
                Objects[i] = NULL;
        }

        g_pInterface = Intr;
        g_pSensorsData = pSensorsData;

        g_pSensorsData->H = 0.5;
        g_pSensorsData->DrainFlowRate = 0;
        g_pSensorsData->InFlowRate = 0;
        g_pSensorsData->OutFlowRate = 0;
        g_pSensorsData->TimeFromBegining = 0;
        g_pSensorsData->DrainSwitch = 0;

        Objects[0] = New CControlObject( ControlObject, g_pSensorsData );

        CMessage msg (CmdUpdate, Interface, ControlObject);
        _SendMessage(msg);

        return RetValue;
}

void CloseModel()
{
        for (int i = 0; i < MAXOBJECT; i++ )
        {
                if ( Objects[i] )
```

```
                {
                        delete Objects[i];
                }
        }
}
```

Створюємо h-файл ControlObject, в якому описуватимемо об'єкт управління, і модифікуємо його наступним чином:

```
#ifndef __CONTROL_OBJECT_
#define __CONTROL_OBJECT_

#include "baseobject.h"
#include "msgcode.h"
#include "SensorData.h"

class CControlObject : public CBaseObject
{
public:
        CControlObject( CModelObject ID, CControlObjectSensors * pSensorsData );
        ~CControlObject();

        double GetRadius();

protected:
        virtual MsgCode Events (MsgCode msg, void * pData);

private:
        // Атрибути
        CControlObjectSensors *m_pSensorData;
        double          m_dTime;        // час моделювання
        double          m_dRadius;      // радіус ємності

        // обробники подій
        MsgCode EvContinue();
        MsgCode EvStart();
        MsgCode EvStop();
        MsgCode EvCalc();
        MsgCode EvChangeParam( MsgCode nCode, void * pData);
        MsgCode EvEmergency( MsgCode nCode, int nNumLine, char* lpFileName );

        // обробники станів
        MsgCode ActionDisactivate();
        MsgCode ActionWaitMsg();
```

```
        MsgCode ActionSetParam (MsgCode nCode, void * pData);
        MsgCode ActionCalc();
        MsgCode ActionEmergency( MsgCode nCode, int nNumLine, char*
lpFileName );
    };

    #endif
```

Створюємо cpp-файл ControlObject, в якому реалізовуватимемо об'єкт управління, і модифікуємо його таким чином:

```
//file ControlObject.cpp


    #include "ControlObject.h"
    #include "basefunction.h"
    #include <math.h>

    CControlObject::CControlObject( CModelObject ID, CControlObjectSensors
*pSensorsData )
                    : CBaseObject(ID)
    {
        m_bInit = TRUE;
        m_pSensorData = pSensorsData;
        m_dTime = 0.1;
        m_dRadius = 0.5;

        ActionDisactivate();
    }
    CControlObject::~CControlObject()
    {
    }
    double CControlObject::GetRadius()
    {
        return m_dRadius;
    }
    MsgCode CControlObject::Events( MsgCode msg, void* pData )
    {
        MsgCode nRetValue = Success;

        switch (msg)
        {
        case CmdChangeInFlowRate:
            {
                double Param = *((double*)pData);
                nRetValue = EvChangeParam( msg, &Param );
```

```
			}
			break;
		case CmdChangeOutFlowRate:
			{
				double Param = *((double*)pData);
				nRetValue = EvChangeParam( msg, &Param );
			}
			break;
		case CmdChangeDrainSwitch:
			{
				BOOL Param = * ((BOOL *) p Data);
				nRetValue = EvChangeParam( msg, &Param );
			}
			break;
		case CmdChangeTime:
			{
				int Param = * ((int *) p Data);
				nRetValue = EvChangeParam( msg, &Param );
			}
			break;
		case CmdStop:
			{
				nRetValue = EvStop();
			}
			break;
		case CmdCalc:
			{
				nRetValue = EvCalc();
			}
			break;
		case CmdStart:
			{
				nRetValue = EvStart();
			}
			break;
		case CmdRadius:
			{
				CMessage msg(CmdRadius, Interface, ControlObject,
&m_dRadius, sizeof(double));
				_SendMessage (msg);
			}
			break;
		default:
			{
```

```
				nRetValue = EvEmergency( ErBadData, __LINE__,
__FILE__ );
			}
			break;
		}
		return nRetValue;
	}
	MsgCode CControlObject::EvContinue()
	{
		MsgCode RetValue = Success;
		switch (m_nState)
		{
		case StSetParam:
		case StCalc:
			{
				RetValue = ActionWaitMsg();
			}
			break;
		case StEmergency:
			{
				RetValue = ActionDisactivate();
			}
			break;
		case StDisactivate:
			{
				// Ігнорується
			}
			break;
		default:
			{
				RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
			}
			break;
		}

		return RetValue;
	}
	MsgCode CControlObject::EvStart()
	{
		MsgCode RetValue = Success;
		switch (m_nState)
		{
		case StDisactivate:
		case StWaitMsg:
```

```
				{
					RetValue = ActionWaitMsg();
				}
				break;
			default:
				{
					RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
				}
				break;
			}

			return RetValue;
		}
		MsgCode CControlObject::EvStop()
		{
			MsgCode RetValue = Success;
			switch (m_nState)
			{
			case StWaitMsg:
				{
					RetValue = ActionDisactivate();
				}
				break;
			case StDisactivate:
			case StEmergency:
				{
					// Ігнорується
				}
				break;
			case StSetParam:
			case StCalc:
				{
					RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
				}
				break;
			}
			return RetValue;
		}
		MsgCode CControlObject::EvCalc()
		{
			MsgCode RetValue = Success;

			switch (m_nState)
```

```
		{
		case StWaitMsg:
			{
				RetValue = ActionCalc();
			}
			break;
		case StDisactivate:
		case StEmergency:
			{
				// Ігнорується
			}
			break;
		case StSetParam:
		case StCalc:
			{
				RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
			}
			break;
		}
		return RetValue;
	}
	MsgCode CControlObject::EvChangeParam( MsgCode nCode, void* pData )
	{
		MsgCode RetValue = Success;

		switch (m_nState)
		{
		case StDisactivate:
		case StWaitMsg:
		case StSetParam:
			{
				RetValue = ActionSetParam(nCode, pData);
			}
			break;
		case StEmergency:
			{
				// Ігнорується
			}
			break;
		case StCalc:
			{
				RetValue = ActionEmergency( ErExecutTPS, __LINE__,
__FILE__ );
			}
```

```
                break;
        }

        return RetValue;
}
MsgCode CControlObject::EvEmergency( MsgCode nCode, int nNumLine,
char* lpFileName )
{
        MsgCode RetValue = Success;
        switch (m_nState)
        {
        case StDisactivate:
        case StEmergency:
                {
                        // Ігнорується
                }
                break;
        case StWaitMsg:
        case StSetParam:
        case StCalc:
                {
                        RetValue = ActionEmergency( nCode, nNumLine,
lpFileName );
                }
                break;
        case CmdStart:
                {
                        RetValue = EvStart();
                }
                break;

        }

        return RetValue;
}

// обробники станів
MsgCode CControlObject::ActionDisactivate()
{
        ChangeState( StDisactivate );

        m_pSensorData->TimeFromBegining = 0;
        m_pSensorData->DrainFlowRate = 0;
        m_pSensorData->InFlowRate = 0;
        m_pSensorData->OutFlowRate = 0;
```

```
        m_pSensorData->H = 0.5;
        CMessage msg (CmdUpdate, Interface, ControlObject);
        _SendMessage (msg);

        return Success;
}
MsgCode CControlObject::ActionWaitMsg()
{
        ChangeState( StWaitMsg );

        return Success;
}
MsgCode CControlObject::ActionSetParam( MsgCode nCode, void* pData )
{
        ChangeState( StSetParam );

        switch (nCode)
        {
        case CmdChangeInFlowRate:
                {
                        m_pSensorData->InFlowRate = *(double*)pData;
                        CMessage msg (CmdUpdate, Interface, ControlObject);
                        _SendMessage (msg);
                }
                break;
        case CmdChangeOutFlowRate:
                {
                        m_pSensorData->OutFlowRate = *(double*)pData;
                        CMessage msg (CmdUpdate, Interface, ControlObject);
                        _SendMessage (msg);
                }
                break;
        case CmdChangeDrainSwitch:
                {
                        m_pSensorData->DrainSwitch = *(BOOL*)pData;
                        CMessage msg (CmdUpdate, Interface, ControlObject);
                        _SendMessage (msg);
                }
                break;
        }
        return EvContinue();
}
MsgCode CControlObject::ActionCalc()
{
        ChangeState( StCalc );
```

```
		if ( m_pSensorData->DrainSwitch )
		{
			double dSpeed = sqrt(2 * 9.81 * m_pSensorData->H);
			m_pSensorData->DrainFlowRate = 3.1415 * pow(0.05, 2) *
dSpeed * m_dTime;
		}
		else
		{
			m_pSensorData->DrainFlowRate = 0;
		}

		// Прийшло в ємність m*m*m
		double Ro = 1000;

		double dInV = (m_pSensorData->InFlowRate * m_dTime) / Ro;
		// Пішло
		double dOutV = m_pSensorData->DrainFlowRate * m_dTime +
			(m_pSensorData->OutFlowRate * m_dTime) / Ro;
		double dV = dInV - dOutV;
		double dH = m_pSensorData->H, dIntV = 0;
		double dDeltaH = 0.00001;
		if ( dV < 0 )
		{
			while ( fabs( dIntV ) < fabs( dV ))
			{
				dIntV += 3.14 * (m_dRadius * m_dRadius - pow(m_dRadius
- m_pSensorData->H, 2)) * dDeltaH;
				m_pSensorData->H = m_pSensorData->H - dDeltaH;
			}
		}
		else
		{
			while ( fabs( dIntV ) < fabs( dV ))
			{
				dIntV += 3.14 * (m_dRadius * m_dRadius - pow(m_dRadius
- m_pSensorData->H, 2)) * dDeltaH;
				m_pSensorData->H = m_pSensorData->H + dDeltaH;
			}
		}
		if ( m_pSensorData->H > m_dRadius ) m_pSensorData->H =
m_dRadius;
		if ( m_pSensorData->H < 0 )
		{
			m_pSensorData->H = 0;
```

```
		}

		m_pSensorData->TimeFromBegining = m_pSensorData->TimeFromBegining + m_dTime;

		CMessage msg (CmdUpdate, Interface, ControlObject);
		_SendMessage(msg);

		return EvContinue();
	}
	MsgCode CControlObject::ActionEmergency( MsgCode nCode, int nNumLine, char* lpFileName )
	{
		ChangeState( StEmergency );

		return EvContinue();
	}
```

Модифікувати h-файл діалогового вікна так:

```
	CMsgManager m_Manager;
	double m_dRadius;
	void UpdateInterface (CMessage &msg);
	void Draw();
```

! Не забудьте підключити MsgManager.h., BaseFunction.h та math.h

У CPP-файлі діалогу оголосити статичну змінну на клас датчиків, а також глобальну константу DELTA_TIME

```
static CControlObjectSensors g_SensorsData;
#define DELTA_TIME 100
```

У обробнику DestroyWindow() викликаємо функцію (CloseModel()), яка закриває модель.

Модифікуємо обробники кнопок так:

```
	void COOControlObjectDlg::OnBtnStart()
	{
		// TODO: Add your control notification handler code here
		CMessage msg (CmdStart, ControlObject, Interface);
		_SendMessage(msg);

		SetTimer(1, DELTA_TIME, NULL);
	}
	void COOControlObjectDlg::OnBtnPause()
	{
		// TODO: Add your control notification handler code here
```

```
        KillTimer(1);
}
void COOControlObjectDlg::OnBtnReset()
{
        // TODO: Add your control notification handler code here
        CMessage msg (CmdStop, ControlObject, Interface);
        _SendMessage(msg);

        KillTimer(1);
        m_slIn.SetPos(0);
        m_slOut.SetPos( 0 );
        m_Open.SetCheck(0);
        m_Auto.SetCheck(0);

        Draw();
}
void COOControlObjectDlg::OnBtnExit()
{
        // TODO: Add your control notification handler code here
        CDialog::OnOK();
}
void COOControlObjectDlg::OnCheckAuto()
{
        // TODO: Add your control notification handler code here
        double dParam;
        char szBuf[16];

        if ( m_Auto.GetCheck() == 1 )
        {
                m_slOut.EnableWindow( FALSE );
                m_slOut.SetPos( m_slIn.GetPos() );

                dParam = m_slOut.GetPos() * 10. / 100.;

                CMessage    msg(    CmdChangeOutFlowRate,    ControlObject,
Interface, &dParam, sizeof( double ));
                _SendMessage(msg);

                sprintf(szBuf, "% g", dParam);
                SetDlgItemText( IDC_OUT_VALUE, szBuf );
        }
        else
        {
                m_slOut.EnableWindow( TRUE );
                dParam = m_slOut.GetPos() * 10. / 100.;
```

```
			CMessage msg( CmdChangeOutFlowRate, ControlObject, Interface, &dParam, sizeof( double ));
			_SendMessage(msg);
			sprintf(szBuf, "% g", dParam);
			SetDlgItemText( IDC_OUT_VALUE, szBuf );

		}
	}
	void COOControlObjectDlg::OnCheckOpen()
	{
		// TODO: Add your control notification handler code here
		BOOL bSwitch;
		if ( m_Open.GetCheck() == 1 )
		{
			bSwitch = 1;
		}
		else
		{
			bSwitch = 0;
		}

		CMessage msg( CmdChangeDrainSwitch, ControlObject, Interface, &bSwitch, sizeof( bool ));
		_SendMessage(msg);
	}
```

Модифікуємо обробники функцій так:

```
	BOOL COOControlObjectDlg::OnInitDialog()
	{
		CDialog::OnInitDialog();

		// Set the icon for this dialog. The framework does this automatically
		// when the application's main window is not a dialog
		SetIcon(m_hIcon, TRUE);			// Set big icon
		SetIcon(m_hIcon, FALSE);		// Set small icon

		// TODO: Add extra initialization here

		m_slIn.SetRange( 0, 100, TRUE );
		m_slOut.SetRange( 0, 100, TRUE );

		m_Open.SetCheck(0);
		m_Auto.SetCheck(0);
```

```
		SetDlgItemText( IDC_IN_VALUE, "0");
		SetDlgItemText( IDC_OUT_VALUE, "0");


		m_SensorList.InsertColumn( 0, "Датчик", LVCFMT_LEFT, 100, -1);
		m_SensorList.InsertColumn( 1, "Показання", LVCFMT_LEFT, 70, 1);

		m_SensorList.InsertItem( 0, "Час");
		m_SensorList.InsertItem( 1, "Вх.расх.");
		m_SensorList.InsertItem( 2, "Вих.расх.");
		m_SensorList.InsertItem( 3, "Рівень");
		m_SensorList.InsertItem( 4, "Злив" );

		m_Manager.SetPointers(this);
		InitModel(&m_Manager, &g_SensorsData);

		CMessage msg (CmdRadius, ControlObject, Interface);
		_SendMessage(msg);

		Draw();
		return TRUE; // return TRUE unless you set the focus to a control
	}
	void COOControlObjectDlg::OnTimer(UINT nIDEvent)
	{
		// TODO: Add your message handler code here and/or call default
		CMessage msg( CmdCalc, ControlObject, Interface );
		_SendMessage(msg);

		Draw();

		CDialog::OnTimer(nIDEvent);
	}
	void COOControlObjectDlg::OnVScroll(UINT nSBCode, UINT nPos,
CScrollBar* pScrollBar)
	{
		// TODO: Add your message handler code here and/or call default
		double dParam;

		CScrollBar* pScroll = (CScrollBar*)GetDlgItem( IDC_SLIDER_IN );
		if ( pScroll == pScrollBar )
		{
			dParam = m_slIn.GetPos() * 10. / 100.;

			CMessage msg( CmdChangeInFlowRate, ControlObject, Interface,
&dParam, sizeof( double ));
```

```
            _SendMessage(msg);

            if ( m_Auto.GetCheck() == 1 )
            {
                m_slOut.SetPos( m_slIn.GetPos() );

                msg.m_nCode=CmdChangeOutFlowRate;
                _SendMessage(msg);
            }
        }
        else
        {
            dParam = m_slOut.GetPos() * 10. / 100.;
            CMessage msg( CmdChangeOutFlowRate, ControlObject,
Interface, &dParam, sizeof( double ));
            _SendMessage(msg);
        }

        CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
    }
```

Реалізовуємо функцію Draw() наступним чином:

```
int nBorder = 10;
CWnd * pWnd = GetDlgItem (IDC_DRAW);
CClientDC dc(pWnd);
CRect rect;
pWnd->GetClientRect(&rect);

CRect DrawRect;
DrawRect.left = rect.left + nBorder;
DrawRect.top=rect.top+nBorder;
DrawRect.right = rect.right - nBorder;
DrawRect.bottom = rect.bottom - nBorder;

int nWidth = rect.right – rect.left;
int nHeight = rect.bottom – rect.top;

CBrush erazeBr(::GetSysColor( COLOR_MENU ));
CBrush Water(RGB(200, 200, 255));

CBitmap Source;
Source.CreateCompatibleBitmap( &dc, nWidth, nHeight );

CDC memDC;
memDC.CreateCompatibleDC(&dc);
```

```
memDC.SelectObject(&Source);

memDC.SelectObject(&erazeBr);
memDC.Rectangle(&rect);
memDC.SelectObject(&Water);

memDC.Arc(      DrawRect,
                CPoint(     nBorder     +     DrawRect.left,
DrawRect.CenterPoint().y ),
                CPoint(nBorder      +      DrawRect.right,
DrawRect.CenterPoint().y));

double dScaleY = (DrawRect.Height() / 2.) / m_dRadius;
double dScaleX = (DrawRect.Width() / 2.) / m_dRadius;
int nLevel = int( dScaleY *( m_dRadius - g_SensorsData.H)); /*( m_pObject-
>GetRadius() - g_SensorsData.H )*/

memDC.MoveTo( DrawRect.left, DrawRect.CenterPoint().y );
memDC.LineTo( DrawRect.right, DrawRect.CenterPoint().y );

int x = int( dScaleX * sqrt( pow( m_dRadius, 2 ) - pow( m_dRadius -
g_SensorsData.H, 2 )));
memDC.MoveTo( DrawRect.CenterPoint().x - x, DrawRect.CenterPoint().y +
nLevel );
memDC.LineTo( DrawRect.CenterPoint().x + x, DrawRect.CenterPoint().y +
nLevel );
if ( nLevel < ( DrawRect.Height() / 2. - 1 ))
{
    memDC.FloodFill( DrawRect.CenterPoint().x, DrawRect.bottom - 2, 0 );
}
dc.BitBlt( 0, 0, nWidth, nHeight, &memDC, 0, 0, SRCCOPY );
memDC.DeleteDC();
```

Реалізовуємо функцію UpdateInterface (CMessage&msg) наступним чином:

```
char szBuf[32];
switch ( msg.m_nCode )
{
case CmdUpdate:
    {
        sprintf(szBuf, "% g", g_SensorsData.TimeFromBegining);
        m_SensorList.SetItem( 0, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

        sprintf(szBuf, "% g", g_SensorsData.InFlowRate);
        SetDlgItemText( IDC_IN_VALUE, szBuf );
```

Translated from Russian to Ukrainian - www.onlinedoctranslator.com


```
			m_slIn.SetPos( ceil(g_SensorsData.InFlowRate * 100 / 10. ));
			m_SensorList.SetItem( 1, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

			sprintf(szBuf, "% g", g_SensorsData.OutFlowRate);
			SetDlgItemText( IDC_OUT_VALUE, szBuf );
			m_slOut.SetPos( ceil(g_SensorsData.OutFlowRate * 100 / 10. ));
			m_SensorList.SetItem( 2, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

			sprintf(szBuf, "% g", g_SensorsData.H);
			m_SensorList.SetItem( 3, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

			sprintf(szBuf, "% g", g_SensorsData.DrainFlowRate);
			m_SensorList.SetItem( 4, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
		}
		break;
	case CmdRadius:
		{
			m_dRadius = *((double*)msg.m_pData);
		}
		break;
	}
```

ЛАБОРАТОРНА РОБОТА №7

Мета роботи: розробити повністю об'єктно-орієнтовану модель стенду та реалізувати управління з відхилення.

У цій лабораторній роботі за основу беремо Лабораторну роботу №6, але будемо регулювати за відхиленням регульованої величини.

Завдання лабораторного практикуму:

1. Розробити об'єктно-орієнтовану модель стенду.
2. Реалізувати модель у програмному коді.
3. Провести тестування.

Теоретичні відомості

У лабораторній роботі №6 було показано, що спроба керувати рівнем рідини в резервуарі, використовуючи закон збереження маси (скільки води влили, стільки ж мають вилити) не працездатна у реальному житті. Завжди знайдуться якісь невраховані витоки, що порушують баланс. Тому для підтримки заданого рівня води у посудині необхідно використовувати інший принцип.

Такий принцип існує і називається управлінням з відхилення. Перш ніж ми його розглянемо, необхідно згадаємо звичайну металеву пружину або канцелярську гумку. Якщо їх деформувати, вони завжди повертаються у вихідний стан. Чим більша деформація, тим більша сила, що її викликає. З курсу фізика ви знаєте, що ця сила визначається виразом:

$$F = -kx \tag{1}$$

де x - Величина деформації;
k - Коефіцієнт пружності.

Допустимо тепер, що x - Це відхилення регульованої величини (рівня рідини в ємності) від заданого рівня. Тоді F - Це величина, на яку треба змінити керуючий вплив (витрата рідини на виході ємності).

Такий підхід і називається управлінням з відхилення. Теоретично автоматичного управління величина позначається символом G_B, величина - символом ε, А вираз (1) називають пропорційним законом управління або П-законом управління.

Розробка моделі

Модифікуємо модель нашого навчального стенду таким чином, щоб з'явилася можливість керувати рівнем відхилення води. Тепер в автоматичному режимі величина витрати буде визначатися, виходячи з поточного відхилення рівня води від заданого значення.

За основу візьмемо проект лабораторної роботи №6. Введемо в модель ще один об'єкт – систему управління (СУ). СУ відповідатиме за моделювання керуючих впливів для ОУ. У ній буде розраховуватись поточне значення вихідного потоку рідини.

Клас, що описує СУ, так само як і клас ОУ успадковуватиме властивості базового класу.

У даному лабораторному практикумі необхідно зробити інтерфейс користувача, ОУ і СУ незалежними один від одного.

Визначимося з атрибутами СУ. Виходячи з досвіду попередньої роботи, атрибутами СУ будуть вказівники на:

- датчик режиму (ручний/автоматичний);
- Датчик коефіцієнта посилення;
- Датчик зони нечутливості;
- Датчик зони перерегулювання;
- Датчик заданого рівня.

Інформаційна модель набуде вигляду:

Мал. 7.1 – Інформаційна модель

Модель станів для ОУ залишимо без змін. Для спрощення завдання цю саму модель приймемо і СУ.

Для ОУ використовуємо математичну модель, наведену у лабораторній роботі №3. Для СУ алгоритм розрахунку наведено нижче.

Приймемо, що відхилення регульованої величини (у разі рівня води) визначається выражением:

$$\varepsilon = h_{зад} - h_i \tag{2}$$

де $h_{зад}$ - Заданий рівень;

h_i - поточний рівень.

Приймемо, що керувати рівнем води ми будемо шляхом зміни вихідної витрати G_B.

Відповідно до (1) зміна керуючого впливу визначається як:

$$\mu = k\varepsilon \tag{3}$$

Відповідно нове значення вихідної витрати G_B визначатиметься як:

$$G_A = G_A \pm \mu \tag{4}$$

Знак $\pm$ в (4) говорить про те, що в залежності від аналізованого процесу, що управляє вплив має або зменшуватися, або збільшуватися.

Розглянемо наш випадок. Припустимо, що поточний рівень менший за заданий. Тоді величини ε, а, отже, і μ позитивні. Для того, щоб підвищити рівень води необхідно зменшити витрату G_B. Отже, (4) повинен стояти знак мінус. Аналогічні висновки для випадку, коли керуючим впливом є вхідна витрата G_A на рівняння (4) повинен стояти знак плюс.

Специфікація функцій класу CBaseObject буде ідентичною, розробленою в лабораторній роботі №6.

Специфікацію функцій класу CControlObject залишимо без змін т.к. модель станів не змінювалася. Така сама специфікація буде і для класу CControlSystem. Це, як ви розумієте, тим, що моделі станів для ОУ і СУ подібні.

Програмна реалізація

Інтерфейс користувача візьмемо з попередньої лабораторної роботи. Для цього необхідно уважно акуратно виконати такі операції.

Після створення проекту на екрані з'явиться заготівля діалогового вікна, його потрібно закрити.

Після цього в меню File/Open вкажіть файл OOControlObject.rc, який знаходиться у папці проекту попереднього лабораторного практикуму. У полі «Open as» виберіть пункт Text та натисніть кнопку Open. Необхідно знайти ділянку наступного коду:

```
IDD_OOCONTROLOBJECT_DIALOG DIALOGEX 0, 0, 448, 255
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "OOControlObject"
FONT 8, "MS Sans Serif"
BEGIN
 CONTROL "", IDC_DRAW, "Static", SS_BLACKFRAME,47,7,153,167
 CONTROL "Slider1", IDC_SLIDER_IN, "msctls_trackbar32",
TBS_AUTOTICKS | TBS_VERT | WS_TABSTOP,14,49,23,102
 CONTROL "Slider1", IDC_SLIDER_OUT, "msctls_trackbar32",
TBS_AUTOTICKS | TBS_VERT | TBS_TOP | WS_TABSTOP,212,49,
 22,102
 LTEXT "Static", IDC_IN_VALUE, 16,186,19,8
 LTEXT "Static", IDC_OUT_VALUE, 218,186,19,8
 PUSHBUTTON "Start",IDC_BTN_START,285,194,50,14
 PUSHBUTTON "Pause",IDC_BTN_PAUSE,353,194,50,14
 PUSHBUTTON "Reset", IDC_BTN_RESET, 284,220,50,14
 PUSHBUTTON "Exit",IDC_BTN_EXIT,353,222,50,14
 CONTROL "Open", IDC_CHECK_OPEN, "Button", BS_AUTOCHECKBOX |
 WS_TABSTOP,272,16,33,10
 CONTROL "Auto", IDC_CHECK_AUTO, "Button", BS_AUTOCHECKBOX |
 WS_TABSTOP,271,30,31,10
 LTEXT "Величина вхідного потоку", IDC_STATIC,7,21,35,26
 LTEXT "Величина вихідного потоку", IDC_STATIC,204,20,45,29
 CONTROL "List1", IDC_LIST, "SysListView32", LVS_REPORT |
LVS_NOSORTHEADER | WS_BORDER | WS_TABSTOP,283,59,120,119
END
```

Також відкрийте файл ModelASU.rc, знайдіть у ньому аналогічну секцію, яка буде виглядати наступним чином:

```
/////////////////////////////////////////////// ///////////////////////////
//
// Dialog
//

IDD_ MODELASU _DIALOG DIALOGEX 0, 0, 320, 200
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "ModelASU"
FONT 8, "MS Sans Serif"
BEGIN
 DEFPUSHBUTTON "OK", IDOK, 260,7,50,14
 PUSHBUTTON "Cancel",IDCANCEL,260,23,50,14
LTEXT "TODO: Place dialog controls here.",IDC_STATIC,50,90,200,8
END
```

Тепер копіюємо вміст OOControlObject.rc у ModelASU.rc, не забуваючи змінити назви проекту. Відкриваємо файл Resourche.h попереднього лабораторного практикуму та копіюємо у наш Resourche.h.

Інтерфейс користувача набуде вигляду як показано на рис. 7.2, на якому також зображено:

- Лист (ListBox): IDC_PROTO.

Привласнюємо ідентифікатор змінної

Ідентифікатор	**Тип**	**Ім'я змінної**
IDC_PROTO	CListCtrl	m_Proto

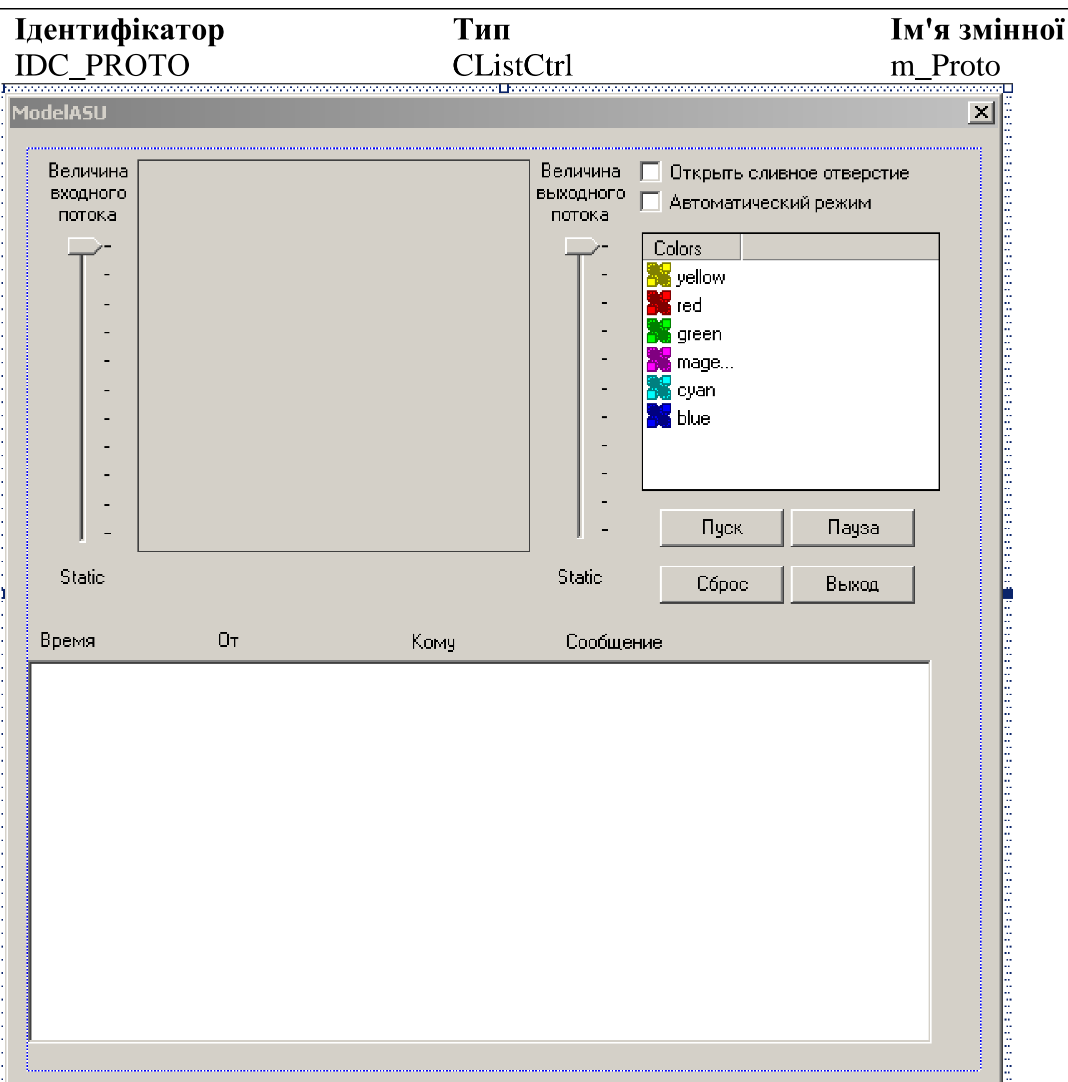


Рисунок 7.2 – Інтерфейс користувача

Необхідно створити змінні, що дозволяють керувати вмістом осередків:

Поле	Ідентифікатор змінної
Member Variable name	m_Auto
Category	Control

Поле	Ідентифікатор змінної
Member Variable name	m_Open
Category	Control

Поле	Ідентифікатор змінної
Member Variable name	m_SensorList
Category	Control

Поле	Ідентифікатор змінної
Member Variable name	m_slln
Category	Control

Поле	Ідентифікатор змінної
Member Variable name	m_slOut
Category	Control

Поле	Ідентифікатор змінної
Member Variable name	m_Proto
Category	Control

Створюємо обробники кнопок OnBtnStart(), OnBtnPause(), OnBtnReset(), OnBtnExit(), OnCheckAuto(), OnCheckOpen() двічі клацнувши на кнопці мишкою. Також створити обробники OnInitDialog(), OnTimer(), OnVScroll().

Приєднуємо до проекту файли MsgManager.h, MsgManager.cpp, IMsgReceive.h, SensorData.h, ControlObject.h, BaseObject.cpp, basefunction.h, basefunction.cpp. Нагадуємо, що файли необхідно спочатку фізично скопіювати до папки з поточним проектом, а вже після цього підключати до проекту.

! Якщо немає жодних застережень щодо тексту, значить, файли залишаємо без змін.

У файлі BaseObject.h в атрибутах доступу базового класу вносимо деякі зміни:

```
virtual MsgCode Events( CMessage& Msg ) = 0;
```

Код файлу basefunction.cpp набуде вигляду:

```
#include "BaseFunction.h"
#include "IMsgReceive.h"
#include "SensorData.h"
#include "ControlObject.h"
#include "ControlSystem.h"

#define MAXOBJECT 4
// Objects[0] - об'єкт управління
// Objects[1] - система управління
static CBaseObject *Objects [MAXOBJECT];

// Інтерфейсний клас
static IMsgReceive * g_pInterface = NULL;

CSensorsData *g_pSensorsData = NULL;

// Надіслати повідомлення об'єкту ПЗ.
// Msg - нове повідомлення
// Повернення: Success – нормальне завершення, інше – код завершення.
// функція шукає у списку об'єкт в потрібний і передає йому повідомлення.
MsgCode _SendMessage( CMessage& Msg )
```

```
{
MsgCode RetValue = Success;

if ( Msg.m_nObjectTo == Interface )
{
       RetValue=g_pInterface->ReceiveMessage( Msg );
}
else if ( Msg.m_nObjectTo == AnyObject )
{
       for (int i = 0; i < MAXOBJECT; i++ )
       {
              if ( Objects[i] )
              {
                     RetValue = Objects[i]->ProcessMessage( Msg );
              }
       }
}
else
{
       for (int i = 0; i < MAXOBJECT; i++ )
       {
              if (( Objects[i] ) && ( Objects[i]->m_nIDObject == Msg.m_nObjectTo ))
              {
                     RetValue = Objects[i]->ProcessMessage( Msg );
              }
       }
}
return RetValue;
}

MsgCode InitModel( IMsgReceive *Intr, CSensorsData *pSensorsData )
{
MsgCode RetValue = Success;
if ( !Intr || !pSensorsData ) return ObjectNotInitiate;
for (int i = 0; i < MAXOBJECT; i++ )
{
       Objects[i] = NULL;
}
g_pInterface = Intr;
g_pSensorsData = pSensorsData;
g_pSensorsData->msControlObject.H = 0.5;
g_pSensorsData->msControlObject.DrainFlowRate = 0;
g_pSensorsData->msControlObject.InFlowRate = 0;
g_pSensorsData->msControlObject.OutFlowRate = 0;
g_pSensorsData->msControlObject.TimeFromBegining = 0;
g_pSensorsData->msControlObject.DrainSwitch = 0;

g_pSensorsData->msControlSystem.pH = &( g_pSensorsData->msControlObject.H );
g_pSensorsData->msControlSystem.pOutFlowRate      =      &(      g_pSensorsData-
>msControlObject.OutFlowRate );

Objects[0] = new CControlObject( ControlObject, &( g_pSensorsData->msControlObject
));
Objects[1] = new CControlSystem( ControlSystem, &( g_pSensorsData->msControlSystem
));

CMessage msg (CmdUpdate, Interface, Interface);
_SendMessage(msg);
return RetValue;
}

void CloseModel()
```

```
{
for (int i = 0; i < MAXOBJECT; i++ )
{
	if ( Objects[i] )
	{
		delete Objects[i];
	}
}
}

void GetIDText( MsgCode ID, char *pText)
{
switch (ID)
{
	case EvAutoModeOn:		sprintf (pText, "Увімкнено автоматичний режим");
break;
	case EvAutoModeOff:		sprintf (pText, "Вимкнено автоматичний режим");
break;
	case EvDrainSwitchOn:	sprintf (pText, "Відкрито зливний отвір"); break;
	case EvDrainSwitchOff:	sprintf (pText, "Закрито зливний отвір"); break;
	default:				sprintf (pText, "Unknown"); break;
}
}


void GetIDText( CModelObject ID, char *pText)
{
switch (ID)
{
	case ControlObject:		sprintf (pText, "Об'єкт управління"); break;
	case ControlSystem:		sprintf (pText, "Система керування"); break;
	case Interface:			sprintf (pText, "Інтерфейс"); break;
	default:				sprintf (pText, "Unknown"); break;
}
}
```

Створюємо h-файл MsgCode.h і модифікуємо його так:

```
#ifndef _MSGCODE_H_
#define _MSGCODE_H_

enum MsgCode
{
Success				= 0,
// Команди
CmdStart			= 1,
CmdStop				= 2,
CmdCalc				= 3,
CmdUpdate			= 4,
CmdReset			= 5,
CmdSetLevel			= 6,
CmdSetOutFlowRate	= 7,
// Стан
StDisactivate		= 100,
StWaitMsg			= 101,
StSetParam			= 102,
StCalc				= 103,
StEmergency			= 104,
// Події
EvAutoModeOn		= 200,
```

```
EvAutoModeOff       = 201,
EvDrainSwitchOn           = 202,
EvDrainSwitchOff    = 203,
EvInFlowRate        = 204,
EvOutFlowRate       = 205,
EvChangeH                 = 206,
EvChangeTime        = 207,
EvChangeGain        = 208,
EvChangeSens        = 209,
EvChangeOverCor           = 210,
EvCangeLevel        = 211,
// Помилки
ObjectNotInitiate   = 1000,
ErExecutTPS               = 1001,
ErBadData                 = 1002,
};

#endif
```

Створюємо h-файл defines.h і модифікуємо його так:

```
#ifndef __DEFINES_
#define __DEFINES_
#include "MsgCode.h"
#include "stdafx.h"

enum CModelObject
{
// Об'єкти ПЗ
NotDefined = 10000,
AnyObject = 10001,
Interface = 10002,
ControlObject = 10003,
ControlSystem = 10004,
};

struct CMessage
{
MsgCode m_nCode;
CModelObject m_nObjectFrom;
CModelObject m_nObjectTo;
void * m_pData;
CMessage()
{
      m_nObjectFrom = NotDefined;
      m_nObjectTo=NotDefined;
      m_nCode = Success;
      m_pData = NULL;
};
CMessage( MsgCode nMsg, CModelObject nObjectFrom = NotDefined, CModelObject
nObjectTo = AnyObject ) // спеціальна функція, що передає код повідомлення
{
      m_nObjectFrom = nObjectFrom;
      m_nObjectTo=nObjectTo;
      m_nCode = nMsg;
      m_pData = NULL;
};

CMessage( MsgCode nMsg, CModelObject nObjectFrom, CModelObject nObjectTo, void*
pData, int nSize ) // спеціальна функція, яка передає код повідомлення від кого і кому,
дані та розмір
```

```
{
        m_nObjectFrom = nObjectFrom;
        m_nObjectTo=nObjectTo;
        m_nCode = nMsg;
        m_pData = pData;
};
};
#endif
```

Змінюємо ім'я структури struct CcontrolObjectSensors на struct ControlObjectSensors у файлі SensorData.h, а також оголошуємо дві нові структури:

```
struct ControlSystemSensors
{
BOOL   bIsOn; //увімкнено автоматичний режим
double K; //коефіцієнт посиленнярегулятора
double SensitivityZone; //Зона нечутливості
double overcorrectionZone; //Зонаперерегулювання
double AssignLevel; //Заданий рівень
double *pH; // рівень води у резервуарім
double *pOutFlowRate; // Витрата води на виході кг/з
};

struct CSensorsData
{
ControlObjectSensors msControlObject;
ControlSystemSensors msControlSystem;
};
```

Код файлу MsgManager.cpp набуде вигляду:

```
#include "stdafx.h"
#include "MsgManager.h"
#include "Resource.h"
#include "ModelASUDlg.h"


CMsgManager::CMsgManager()
{
m_pDlg = NULL;
}

void CMsgManager::SetPointers( CModelASUDlg* pDlg )
{
m_pDlg = pDlg;
}

/ / Функція обробки повідомлення.
// Параметри:
// Msg – посилання на клас повідомлення.
// Повернення: Success - нормальне завершення, інше код помилки.
MsgCode CMsgManager::ReceiveMessage(CMessage& Msg)
{
m_pDlg->UpdateInterface( Msg );
m_pDlg->Draw();

return Success;
}
```

У файлі BaseFunction.h змінюється тип вказівника на змінну pSensorsData на CSensorsData і оголошуємо дві функції:

```
void GetIDText(MsgCode ID, char *pText);
void GetIDText( CModelObject ID, char * pText);
```

Створюємо cpp-файл та модифікуємо його наступним чином ControlObject.cpp:

```
#include "ControlObject.h"
#include "basefunction.h"
#include <math.h>

CControlObject::CControlObject(    CModelObject    ID,    ControlObjectSensors
*pSensorsData )
: CBaseObject(ID)
{
m_bInit = TRUE;
m_pSensorsData = pSensorsData;
m_dTime = 0.1;
m_dRadius = 0.5;
ActionDisactivate();
}

CControlObject::~CControlObject()
{
}

double CControlObject::GetRadius()
{
return m_dRadius;
}

MsgCode CControlObject::Events( CMessage& Msg )
{
MsgCode nRetValue = Success;
switch (Msg.m_nCode)
{
	case EvInFlowRate:
	{
		double Param = *((double*)Msg.m_pData);
		nRetValue = EvChangeParam( Msg.m_nCode, &Param );
	}
	break;
	case EvOutFlowRate:
	{
		double Param = *((double*)Msg.m_pData);
		nRetValue = EvChangeParam( Msg.m_nCode, &Param );
	}
	break;
	case EvDrainSwitchOn:
	case EvDrainSwitchOff:
	{
		nRetValue = EvChangeParam( Msg.m_nCode, NULL );
	}
	break;
	case EvChangeTime:
	{
		int Param = * ((int *) Msg.m_pData);
		nRetValue = EvChangeParam( Msg.m_nCode, &Param );
	}
	break;
	case CmdStop:
```

```
		{
			nRetValue = EvStop();
		}
		break;
		case CmdCalc:
		{
			nRetValue = EvCalc();
		}
		break;
		case CmdStart:
		{
			nRetValue = EvStart();
		}
		break;
		default:
		{
			nRetValue = EvEmergency( ErBadData, __LINE__, __FILE__ );
		}
		break;
	}
	return nRetValue;
	}

	//обробники подій
	MsgCode CControlObject::EvContinue()
	{
	MsgCode RetValue = Success;
	switch (m_nState)
	{
		case StSetParam:
		case StCalc:
		{
			RetValue = ActionWaitMsg();
		}
		break;
		case StEmergency:
		{
			RetValue = ActionDisactivate();
		}
		break;
		case StDisactivate:
		{
			// Ігнорується
		}
		break;
		default:
		{
			RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
		}
		break;
	}
	return RetValue;
	}

	MsgCode CControlObject::EvStart()
	{
	MsgCode RetValue = Success;
	switch (m_nState)
	{
		case StDisactivate:
		case StWaitMsg:
		{
```

```
			RetValue = ActionWaitMsg();
		}
		break;
		default:
		{
			RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
		}
		break;
	}
	return RetValue;
	}

	MsgCode CControlObject::EvStop()
	{
	MsgCode RetValue = Success;
	switch (m_nState)
	{
		case StWaitMsg:
		{
			RetValue = ActionDisactivate();
		}
		break;
		case StDisactivate:
		case StEmergency:
		{
			// Ігнорується
		}
		break;
		case StSetParam:
		case StCalc:
		{
			RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
		}
		break;
	}
	return RetValue;
	}

	MsgCode CControlObject::EvCalc()
	{
	MsgCode RetValue = Success;
	switch (m_nState)
	{
		case StWaitMsg:
		{
			RetValue = ActionCalc();
		}
		break;
		case StDisactivate:
		case StEmergency:
		{
			// Ігнорується
		}
		break;
		case StSetParam:
		case StCalc:
		{
			RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
		}
		break;
	}
	return RetValue;
```

```
}

MsgCode CControlObject::EvChangeParam( MsgCode nCode, void* pData )
{
MsgCode RetValue = Success;
switch (m_nState)
{
	case StDisactivate:
	case StWaitMsg:
	case StSetParam:
	{
		RetValue = ActionSetParam(nCode, pData);
	}
	break;
	case StEmergency:
	{
		// Ігнорується
	}
	break;
	case StCalc:
	{
		RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
	}
	break;
}
return RetValue;
}

MsgCode CControlObject::EvEmergency( MsgCode nCode, int nNumLine, char* lpFileName
)
{
MsgCode RetValue = Success;
switch (m_nState)
{
	case StDisactivate:
	case StEmergency:
	{
		// Ігнорується
	}
	break;
	case StWaitMsg:
	case StSetParam:
	case StCalc:
	{
		RetValue = ActionEmergency( nCode, nNumLine, lpFileName );
	}
	break;
	case CmdStart:
	{
		RetValue = EvStart();
	}
	break;
}
return RetValue;
}

// обробники станів
MsgCode CControlObject::ActionDisactivate()
{
ChangeState( StDisactivate );
m_pSensorsData->TimeFromBegining = 0;
m_pSensorsData->DrainFlowRate = 0;
```

```
m_pSensorsData->InFlowRate = 0;
m_pSensorsData->OutFlowRate = 0;
m_pSensorsData->H = 0.5;

CMessage msg (CmdUpdate, ControlObject, Interface);
_SendMessage (msg);

return Success;
}

MsgCode CControlObject::ActionWaitMsg()
{
ChangeState( StWaitMsg );
return Success;
}

MsgCode CControlObject::ActionSetParam( MsgCode nCode, void* pData )
{
ChangeState( StSetParam );
switch (nCode)
{
	case EvInFlowRate:
	{
		m_pSensorsData->InFlowRate = *(double*) pData;
		CMessage msg (CmdUpdate, ControlObject, Interface);
		_SendMessage (msg);
	}
	break;
	case EvOutFlowRate:
	{
		m_pSensorsData->OutFlowRate = *(double*) pData;
		CMessage msg (CmdUpdate, ControlObject, Interface);
		_SendMessage (msg);
	}
	break;
	case EvDrainSwitchOn:
	{
		m_pSensorsData->DrainSwitch = TRUE;
		CMessage msg(EvDrainSwitchOn, ControlObject, Interface);
		_SendMessage (msg);
	}
	break;
	case EvDrainSwitchOff:
	{
		m_pSensorsData->DrainSwitch = FALSE;
		CMessage msg(EvDrainSwitchOn, ControlObject, Interface);
		_SendMessage (msg);
	}
	break;
}
return EvContinue();
}

MsgCode CControlObject::ActionCalc()
{
ChangeState( StCalc );
if ( m_pSensorsData->DrainSwitch )
{
	double dSpeed = sqrt(2 * 9.81 * m_pSensorsData->H);
	m_pSensorsData->DrainFlowRate = 3.1415* pow( 0.05, 2 ) * dSpeed *
	m_dTime;
}
```

```
	else
	{
		m_pSensorsData->DrainFlowRate = 0;
	}
	// Надійшло в ємність m*m*m
	double Ro = 1000;
	double dInV = (m_pSensorsData->InFlowRate * m_dTime) / Ro;
	// Пішло
	double dOutV = m_pSensorsData->DrainFlowRate * m_dTime + (m_pSensorsData->OutFlowRate * m_dTime ) / Ro;
	double dV = dInV - dOutV;
	double dH = m_pSensorsData->H, dIntV = 0;
	double dDeltaH = 0.00001;
	if ( dV < 0 )
	{
		while ( fabs( dIntV ) < fabs( dV ))
		{
			dIntV += 3.14 * (m_dRadius * m_dRadius - pow(m_dRadius - m_pSensorsData->H, 2)) * dDeltaH;
			m_pSensorsData->H = m_pSensorsData->H - dDeltaH;
		}
	}
	else
	{
		while ( fabs( dIntV ) < fabs( dV ))
		{
			dIntV += 3.14 * (m_dRadius * m_dRadius - pow(m_dRadius - m_pSensorsData->H, 2)) * dDeltaH;
			m_pSensorsData->H = m_pSensorsData->H + dDeltaH;
		}
	}
	if ( m_pSensorsData->H > m_dRadius ) m_pSensorsData->H = m_dRadius;
	if ( m_pSensorsData->H < 0 )
	{
		m_pSensorsData->H = 0;
	}
	m_pSensorsData->TimeFromBegining = m_pSensorsData->TimeFromBegining + m_dTime;

	CMessage msg (CmdUpdate, ControlObject, Interface);
	_SendMessage(msg);

	return EvContinue();
	}

	MsgCode CControlObject::ActionEmergency( MsgCode nCode, int nNumLine, char* lpFileName )
	{
	ChangeState( StEmergency );
	return EvContinue();
	}
```

У h-файлі діалогу підключаємо файл MsgManager.h, а в класі діалогу оголошуємо наступні функції та атрибути:

```
	CMsgManager  m_Manager;
	void Draw();
	void UpdateInterface( CMessage& msg );
```

У CPP-файлі діалогу підключаємо файли basefunction.h, MsgManager.h, і бібліотеку math.h, також оголошуємо глобальні змінні:

```
#define OBJECT_DELTA_TIME 100
#define SYSTEM_DELTA_TIME 100

#define TIMER_FOR_OBJECT 1
#define TIMER_FOR_SYSTEM 2
static CSensorsData g_SensorsData;
```

Модифікуємо обробники функцій так:

```
BOOL CModelASUDlg::OnInitDialog()
{

CDialog::OnInitDialog();

// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX <0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
	CString strAboutMenu;
	strAboutMenu.LoadString(IDS_ABOUTBOX);
	if (!strAboutMenu.IsEmpty())
	{
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	}
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);			// Set big icon
SetIcon(m_hIcon, FALSE);		// Set small icon

// TODO: Add extra initialization here
// встановлюємо діапазони для слайдерів
m_slIn.SetRange( 0, 100, TRUE );
m_slOut.SetRange( 0, 100, TRUE );

// ініціалізуємо менеджер виведення повідомлень в інтерфейс
m_Manager.SetPointers(this);
m_SensorList.InsertColumn( 0, "Датчик", LVCFMT_LEFT, 100, -1);
m_SensorList.InsertColumn( 1, "Показання", LVCFMT_LEFT, 70, 1);
m_SensorList.InsertItem( 0, "Вх.расх.");
m_SensorList.InsertItem( 1, "Вих.расх.");
m_SensorList.InsertItem( 2, "Рівень");
m_SensorList.InsertItem( 3, "Злив");
m_SensorList.InsertItem( 4, "Авт.реж");
m_SensorList.InsertItem( 5, "Час");
m_SensorList.InsertItem( 6, "Коеф.усил.");
m_SensorList.InsertItem( 7, "Зона нечув.");
m_SensorList.InsertItem( 8, "Зона перерег.");
m_SensorList.InsertItem( 9, "Зад. рівень");
// ініціалізуємо модель
InitModel(&m_Manager, &g_SensorsData);
return TRUE; // return TRUE unless you set the focus to a control
}
```

```
BOOL CModelASUDlg::DestroyWindow()
{
// TODO: Add your specialized code here and/or call the base class
CMessage msg (CmdStop, Interface, AnyObject);
_SendMessage(msg);
CloseModel();
return CDialog::DestroyWindow();
}
```

```
void CModelASUDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar )
{
// TODO: Add your message handler code here and/or call default
double dData;
CScrollBar* pScroll = (CScrollBar*) GetDlgItem (IDC_SLIDER_IN);
if ( pScroll == pScrollBar )
{
	dData = m_slIn.GetPos() * 10. / 100.;
	CMessage msg( EvInFlowRate, Interface, ControlObject, &dData,
	sizeof( double ));
	_SendMessage(msg);
}
else
{
	dData = m_slOut.GetPos() * 10. / 100.;
	CMessage msg( EvOutFlowRate, Interface, ControlObject, &dData,
	sizeof( double ));
	_SendMessage(msg);
}
CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

```
void CModelASUDlg::Draw()
{
int nBorder = 10;
CWnd* pWnd = GetDlgItem(IDC_DRAW);
CClientDC dc(pWnd);
CRect rect;
pWnd->GetClientRect(&rect);
CRect DrawRect;

DrawRect.left = rect.left + nBorder;
DrawRect.top=rect.top+nBorder;
DrawRect.right = rect.right - nBorder;
DrawRect.bottom = rect.bottom - nBorder;
int nWidth = rect.right – rect.left;
int nHeight = rect.bottom – rect.top;
CBrush erazeBr(::GetSysColor( COLOR_MENU ));
CBrush Water (RGB (200, 200, 255));
CBitmap Source;
Source.CreateCompatibleBitmap( &dc, nWidth, nHeight );

CDC memDC;
memDC.CreateCompatibleDC(&dc);
```

```
memDC.SelectObject(&Source);
memDC.SelectObject(&erazeBr);
memDC.Rectangle(&rect);
memDC.SelectObject(&Water);
memDC.Arc( DrawRect,
CPoint(nBorder + DrawRect.left,
DrawRect.CenterPoint().y),
CPoint(nBorder + DrawRect.right,
DrawRect.CenterPoint().y));

double dRadius = 0.5;
double dScaleY = (DrawRect.Height() / 2.) / dRadius;
double dScaleX = (DrawRect.Width() / 2.) / dRadius;
int nLevel = int( dScaleY * ( dRadius - g_SensorsData.msControlObject.H ));
memDC.MoveTo( DrawRect.left, DrawRect.CenterPoint().y );
memDC.LineTo( DrawRect.right, DrawRect.CenterPoint().y );
int x = int( dScaleX * sqrt( pow( dRadius, 2 ) - pow( dRadius -
g_SensorsData.msControlObject.H, 2)));
memDC.MoveTo( DrawRect.CenterPoint().x - x, DrawRect.CenterPoint().y + nLevel );
memDC.LineTo( DrawRect.CenterPoint().x + x, DrawRect.CenterPoint().y + nLevel );
if ( nLevel < ( DrawRect.Height() / 2. - 1 ))
{
	memDC.FloodFill( DrawRect.CenterPoint().x, DrawRect.bottom - 2, 0 );
}
dc.BitBlt( 0, 0, nWidth, nHeight, &memDC, 0, 0, SRCCOPY );
memDC.DeleteDC();
}
```

```
void CModelASUDlg::UpdateInterface( CMessage& msg )
{
char szBuf[512];
switch ( msg.m_nCode )
{
	case CmdUpdate:
	{
		sprintf( szBuf, "% g", g_SensorsData.msControlObject.InFlowRate);
		SetDlgItemText( IDC_IN_VALUE, szBuf );
		m_slIn.SetPos( ceil(g_SensorsData.msControlObject.InFlowRate * 100 /
10. ));
		m_SensorList.SetItem( 0, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

		sprintf( szBuf, "% g", g_SensorsData.msControlObject.OutFlowRate);
		SetDlgItemText( IDC_OUT_VALUE, szBuf );
		m_slOut.SetPos( ceil(g_SensorsData.msControlObject.OutFlowRate * 100
/ 10. ));
		m_SensorList.SetItem( 1, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

		sprintf( szBuf, "% g", g_SensorsData.msControlObject.H);
		m_SensorList.SetItem( 2, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);

		m_Open.SetCheck(g_SensorsData.msControlObject.DrainSwitch);
		if ( g_SensorsData.msControlObject.DrainSwitch )
		{
			m_SensorList.SetItem( 3, 1, LVIF_TEXT, "Відкр.", 0, 0, 0,0);
		}
		else
		{
```

```
                m_SensorList.SetItem( 3, 1, LVIF_TEXT, "Закр.", 0, 0, 0,0);
            }
            m_Auto.SetCheck(g_SensorsData.msControlSystem.bIsOn);
            if ( g_SensorsData.msControlSystem.bIsOn )
            {
                m_SensorList.SetItem( 4, 1, LVIF_TEXT, "Увімк.", 0, 0, 0, 0);
                m_slOut.EnableWindow( FALSE );
            }
            else
            {
                m_SensorList.SetItem( 4, 1, LVIF_TEXT, "Вимк.", 0, 0, 0, 0 );
                m_slOut.EnableWindow( TRUE );
            }
            sprintf(szBuf,                          "%                          g",
g_SensorsData.msControlObject.TimeFromBegining);
            m_SensorList.SetItem( 5, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            sprintf (szBuf, "% g", g_SensorsData.msControlSystem.K);
            m_SensorList.SetItem( 6, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            sprintf(                szBuf,                "%                g",
g_SensorsData.msControlSystem.SensitivityZone);
            m_SensorList.SetItem( 7, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            sprintf(szBuf,                                                   "%g",
g_SensorsData.msControlSystem.OvercorrectionZone);
            m_SensorList.SetItem( 8, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            sprintf (szBuf, "% g", g_SensorsData.msControlSystem.AssignLevel);
            m_SensorList.SetItem( 9, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
        }
        break;
        case EvInFlowRate:
        {
            sprintf(szBuf, "%g", *((double*)msg.m_pData));
            SetDlgItemText( IDC_IN_VALUE, szBuf );
            m_SensorList.SetItem( 0, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
        }
        break;
        case EvOutFlowRate:
        {
            sprintf(szBuf, "%g", *((double*)msg.m_pData));
            SetDlgItemText( IDC_OUT_VALUE, szBuf );
            m_SensorList.SetItem( 1, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            m_slOut.SetPos(
            ceil(g_SensorsData.msControlObject.OutFlowRate * 100/10. ));
        }
        break;
        case EvDrainSwitchOn:
        {
            m_SensorList.SetItem( 3, 1, LVIF_TEXT, "Об'яв", 0, 0, 0, 0 );
            char szText[128];
            GetIDText( msg.m_nCode, szText );
            char szText1[128];
            GetIDText( msg.m_nObjectTo, szText1 );
            char szText2[128];
            GetIDText( msg.m_nObjectFrom, szText2 );
            SYSTEMTIME CurTime;
            GetLocalTime(&CurTime);
            sprintf(   szBuf,   "%2.2d:%2.2d:%2.2d:%3.3d%s%s%s",   CurTime.wHour,
CurTime.wMinute, CurTime.wSecond,
            CurTime.wMilliseconds, szText2, szText1, szText);
            m_Proto.InsertString( 0, szBuf );
        }
        break;
        case EvDrainSwitchOff:
```

```
            {
                    m_SensorList.SetItem( 3, 1, LVIF_TEXT, "Закр.", 0, 0, 0, 0);
                    char szText[128];
                    GetIDText( msg.m_nCode, szText );
                    char szText1[128];
                    GetIDText( msg.m_nObjectTo, szText1 );
                    char szText2[128];
                    GetIDText( msg.m_nObjectFrom, szText2 );
                    SYSTEMTIME CurTime;
                    GetLocalTime(&CurTime);
                    sprintf(   szBuf,   "%2.2d:%2.2d:%2.2d:%3.3d%s%s%s",   CurTime.wHour,
CurTime.wMinute, CurTime.wSecond,
                    CurTime.wMilliseconds, szText2, szText1, szText);
                    m_Proto.InsertString( 0, szBuf );
            }
            break;
            case EvChangeH:
            {
                    sprintf(szBuf, "%g", *((double*)msg.m_pData));
                    m_SensorList.SetItem( 2, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            }
            break;
            case EvChangeTime:
            {
                    sprintf(szBuf, "%g", *((double*)msg.m_pData));
                    m_SensorList.SetItem( 5, 1, LVIF_TEXT, szBuf, 0, 0, 0, 0);
            }
            break;
            case EvAutoModeOn:
            {
                    m_SensorList.SetItem( 4, 1, LVIF_TEXT, "Увімк.", 0, 0, 0, 0);
                    char szText[128];
                    GetIDText( msg.m_nCode, szText );
                    char szText1[128];
                    GetIDText( msg.m_nObjectTo, szText1 );
                    char szText2[128];
                    GetIDText( msg.m_nObjectFrom, szText2 );
                    SYSTEMTIME CurTime;
                    GetLocalTime(&CurTime);
                    sprintf(   szBuf,   "%2.2d:%2.2d:%2.2d:%3.3d%s%s%s",   CurTime.wHour,
CurTime.wMinute, CurTime.wSecond,
                    CurTime.wMilliseconds, szText2, szText1, szText);
                    m_Proto.InsertString( 0, szBuf );
            }
            break;
            case EvAutoModeOff:
            {
                    m_SensorList.SetItem( 4, 1, LVIF_TEXT, "Вимк.", 0, 0, 0, 0 );
                    char szText[128];
                    GetIDText( msg.m_nCode, szText );
                    char szText1[128];
                    GetIDText( msg.m_nObjectTo, szText1 );
                    char szText2[128];
                    GetIDText( msg.m_nObjectFrom, szText2 );
                    SYSTEMTIME CurTime;
                    GetLocalTime(&CurTime);
                    sprintf(   szBuf,   "%2.2d:%2.2d:%2.2d:%3.3d%s%s%s",   CurTime.wHour,
CurTime.wMinute, CurTime.wSecond,
                    CurTime.wMilliseconds, szText2, szText1, szText);
                    m_Proto.InsertString( 0, szBuf );
            }
            break;
```

```
}
}
```

Модифікуємо обробники кнопок так:

```
void CModelASUDlg::OnCheckAuto()
{
// TODO: Add your control notification handler code here
// створюємо команду зупинки та посилаємо її
if ( m_Auto.GetCheck() == 1 )
{
        CMessage      msg(      CmdSetLevel,      Interface,      ControlSystem,
&g_SensorsData.msControlObject.H, sizeof( double ));
        _SendMessage(msg);
        CMessage msg2(EvAutoModeOn, Interface, ControlSystem);
        _SendMessage(msg2);
}
else
{
        CMessage msg(EvAutoModeOff, Interface, ControlSystem);
        _SendMessage(msg);
}
}



void CModelASUDlg::OnBtnExit()
{
// TODO: Add your control notification handler code here
KillTimer (TIMER_FOR_OBJECT);
KillTimer (TIMER_FOR_SYSTEM);
CDialog::OnOK();
}


void CModelASUDlg::OnBtnPause()
{
// TODO: Add your control notification handler code here
// Зупиняємо таймери
KillTimer (TIMER_FOR_OBJECT);
KillTimer (TIMER_FOR_SYSTEM);
}



void CModelASUDlg::OnBtnReset()
{
// TODO: Add your control notification handler code here

KillTimer (TIMER_FOR_OBJECT);
KillTimer (TIMER_FOR_SYSTEM);
// створюємо команду зупинки та посилаємо її
CMessage msg( CmdStop, Interface, AnyObject );
_SendMessage(msg);
// LoadInitData(&g_SensorsData);
msg.m_nCode = CmdReset;
_SendMessage(msg);
}
```

```
void CModelASUDlg::OnBtnStart()
{
// TODO: Add your control notification handler code here
// Створюємо команду запуску та посилаємо її
CMessage msg( CmdStart, Interface, ControlObject );
_SendMessage(msg);
msg.m_nObjectTo=ControlSystem;
_SendMessage(msg);
// Запускаємо таймери для об'єкта та регулятора
SetTimer(TIMER_FOR_OBJECT, OBJECT_DELTA_TIME, NULL);
SetTimer (TIMER_FOR_SYSTEM, SYSTEM_DELTA_TIME, NULL);
}


void CModelASUDlg::OnCheckOpen()
{
// TODO: Add your control notification handler code here
if ( m_Open.GetCheck() == 1 )
{
      CMessage msg(EvDrainSwitchOn, Interface, ControlObject);
      _SendMessage(msg);
}
else
{
      CMessage msg(EvDrainSwitchOff, Interface, ControlObject);
      _SendMessage(msg);
}
}
```

Створюємо h-файл ControlSystem і модифікуємо його так:

```
#ifndef __CONTROLSYSTEM_
#define __CONTROLSYSTEM_

#include "baseobject.h"
#include "msgcode.h"
#include "SensorData.h"
#include <afxwin.h>

class CControlSystem : public CBaseObject
{
public:

CControlSystem( CModelObject ID, ControlSystemSensors * pSensors );
~CControlSystem();
protected:
virtual MsgCode Events(CMessage& Msg);
private:
// Атрибути
ControlSystemSensors *m_pSensors;

// обробники подій
MsgCode EvContinue();
MsgCode EvStart();
MsgCode EvStop();
MsgCode EvCalc();
MsgCode EvChangeParam (MsgCode nCode, void * pData);
MsgCode EvEmergency( MsgCode nCode, int nNumLine, char* lpFileName );

// обробники станів
```

```
MsgCode ActionDisactivate();
MsgCode ActionWaitMsg();
MsgCode ActionSetParam (MsgCode nCode, void * pData);
MsgCode ActionCalc();
MsgCode ActionEmergency( MsgCode nCode, int nNumLine, char *lpFileName );
};
#endif
```

Створюємо cpp-файл та модифікуємо його наступним чином ControlSystem.cpp:

```
#include "ControlSystem.h"
#include "BaseFunction.h"
#include <math.h>

CControlSystem::CControlSystem( CModelObject ID, ControlSystemSensors *pSensors )
: CBaseObject(ID)
{
m_pSensors = pSensors;
ActionDisactivate();
}

CControlSystem::~CControlSystem()
{
}

MsgCode CControlSystem::Events( CMessage& Msg )
{
MsgCode nRetValue = Success;
switch (Msg.m_nCode)
{
	case CmdStart:
	{
		nRetValue = EvStart();
	}
	break;
	case CmdStop:
	{
		nRetValue = EvStop();
	}
	break;
	case CmdCalc:
	{
		if ( m_pSensors->bIsOn )
		{
			nRetValue = EvCalc();
		}
	}
	break;
	case EvChangeGain:
	case EvChangeSens:
	case EvChangeOverCor:
	case EvAutoModeOn:
	case EvAutoModeOff:
	case CmdSetLevel:
	{
		nRetValue = EvChangeParam( Msg.m_nCode, Msg.m_pData );
	}
	break;
	default:
	{
		nRetValue = EvEmergency( ErBadData, __LINE__, __FILE__ );
```

```
        }
        break;
}
return nRetValue;
}

// обробники подій
MsgCode CControlSystem::EvStart()
{
MsgCode RetValue = Success;
switch (m_nState)
{
        case StDisactivate:
        {
                RetValue = ActionWaitMsg();
        }
        break;
        case StWaitMsg:
        {
        }
        break;
        default:
        {
                RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
        }
        break;
}
return RetValue;
}

MsgCode CControlSystem::EvContinue()
{
MsgCode RetValue = Success;
switch (m_nState)
{
        case StSetParam:
        case StCalc:
        {
                RetValue = ActionWaitMsg();
        }
        break;
        case StEmergency:
        {
                RetValue = ActionDisactivate();
        }
        break;
        case StDisactivate:
        {
                // Ігнорується
        }
        break;
        default:
        {
                RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
        }
        break;
}
return RetValue;
}

MsgCode CControlSystem::EvStop()
{
```

```
MsgCode RetValue = Success;
switch (m_nState)
{
	case StWaitMsg:
	{
		RetValue = ActionDisactivate();
	}
	break;
	case StDisactivate:
	case StEmergency:
	{
		// Ігнорується
	}
	break;
	case StSetParam:
	case StCalc:
	{
		RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
	}
	break;
}
return RetValue;
}

MsgCode CControlSystem::EvCalc()
{
MsgCode RetValue = Success;
switch (m_nState)
{
	case StWaitMsg:
	{
		RetValue = ActionCalc();
	}
	break;
	case StDisactivate:
	case StEmergency:
	{
		// Ігнорується
	}
	break;
	case StSetParam:
	case StCalc:
	{
		RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
	}
	break;
}
return RetValue;
}

MsgCode CControlSystem::EvChangeParam( MsgCode nCode, void* pData )
{
MsgCode RetValue = Success;
switch (m_nState)
{
	case StDisactivate:
	case StWaitMsg:
	{
		RetValue = ActionSetParam(nCode, pData);
	}
	break;
	case StEmergency:
```

```
		{
			// Ігнорується
		}
		break;
		case StSetParam:
		case StCalc:
		{
			RetValue = ActionEmergency( ErExecutTPS, __LINE__, __FILE__ );
		}
		break;
	}
	return RetValue;
	}

	MsgCode CControlSystem::EvEmergency( MsgCode nCode, int nNumLine, char* lpFileName
)
	{
	MsgCode RetValue = Success;
	switch (m_nState)
	{
		case StDisactivate:
		case StEmergency:
		{
			// Ігнорується
		}
		break;
		case StWaitMsg:
		case StSetParam:
		case StCalc:
		{
			RetValue = ActionEmergency( nCode, nNumLine, lpFileName );
		}
		break;
	}

	return RetValue;
	}

	// обробники станів
	MsgCode CControlSystem::ActionDisactivate()
	{
	ChangeState( StDisactivate );

	m_pSensors->bIsOn = 0;
	m_pSensors->K = 3;
	m_pSensors->SensitivityZone = 0;
	m_pSensors->OvercorrectionZone = 10;
	m_pSensors->AssignLevel = 0.3;

	return Success;
	}

	MsgCode CControlSystem::ActionWaitMsg()
	{
	ChangeState( StWaitMsg );
	return Success;
	}

	MsgCode CControlSystem::ActionSetParam( MsgCode nCode, void* pData )
	{
	ChangeState( StSetParam );
	switch (nCode)
```

```
{
        case EvChangeGain:
        {
                m_pSensors->K = *(double*) pData;
                CMessage msg( EvChangeGain, ControlSystem, Interface, &m_pSensors->K,
sizeof( double ));
                _SendMessage(msg);
        }
        break;
        case EvChangeSens:
        {
                m_pSensors->SensitivityZone = *(double*) pData;
                CMessage msg( EvChangeSens, ControlSystem, Interface, &m_pSensors-
>SensitivityZone, sizeof( double ));
                _SendMessage(msg);
        }
        break;
        case EvChangeOverCor:
        {
                m_pSensors->OvercorrectionZone = *(double*) pData;
                CMessage msg( EvChangeOverCor, ControlSystem, Interface, &m_pSensors-
>OvercorrectionZone, sizeof( double ));
                _SendMessage(msg);
        }
        break;
        case CmdSetLevel:
        {
                m_pSensors->AssignLevel = *(double*) pData;
                CMessage msg( EvCangeLevel, ControlSystem, Interface, &m_pSensors-
>AssignLevel, sizeof( double ));
                _SendMessage(msg);
        }
        break;
        case EvAutoModeOn:
        {
                m_pSensors->bIsOn = TRUE;
                CMessage msg(EvAutoModeOn, ControlSystem, Interface);
                _SendMessage(msg);
        }
        break;
        case EvAutoModeOff:
        {
                m_pSensors->bIsOn = FALSE;
                CMessage msg(EvAutoModeOff, ControlSystem, Interface);
                _SendMessage(msg);
        }
        break;
}
return EvContinue();
}

MsgCode CControlSystem::ActionCalc()
{
ChangeState( StCalc );
double dDelta = m_pSensors->AssignLevel - *m_pSensors->pH;
double dNewValue = *m_pSensors->pOutFlowRate - m_pSensors->K * dDelta;
if ( dNewValue < m_pSensors->SensitivityZone )
{
        dNewValue = m_pSensors->SensitivityZone;
}
if ( dNewValue > m_pSensors->OvercorrectionZone )
{
```

```
            dNewValue = m_pSensors->OvercorrectionZone;
      }

      CMessage msg( EvOutFlowRate, ControlSystem, ControlObject, &dNewValue, sizeof(
double ));
      _SendMessage(msg);
      return EvContinue();
      }


      MsgCode CControlSystem::ActionEmergency( MsgCode nCode, int nNumLine, char*
lpFileName )
      {
      ChangeState( StEmergency );
      return EvContinue();
      }
```

У файлі baseobject.cpp у функції ProcessMessage змінити ділянку коду на:

```
return Events(Msg);
```

У файлі ControlObject.h в тип доступу protected внести деякі зміни:

```
virtual MsgCode Events(CMessage& Msg);
```

У файлі MsgManager.h змінити діалоговий клас CModelASUDlg.