

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

**О.М. Рисований**

**СИСТЕМНЕ ПРОГРАМУВАННЯ**  
**Графічний інтерфейс користувача (GUI)**

Навчальний посібник

для студентів спеціальностей 123 «Комп'ютерна інженерія»  
та 125 «Кібербезпека»  
вищих навчальних закладів

Рекомендовано вченою радою  
Національного технічного університету  
«Харківський політехнічний інститут»

Харків  
2018

УДК 004.42  
Р54

Рецензенти:

*А.В. Шостак*, канд техн. наук, доц., проф. кафедри комп'ютерних систем та мереж, НАУ «ХАІ»;

*М.Ю. Лосєв*, канд. техн. наук, доц. ХНЕУ ім. Семена Кузнеця

*Рекомендовано вченою радою НТУ «ХПІ» як навчальний посібник для студентів спеціальностей 123 – «Комп'ютерна інженерія», 125 – «Кібербезпека», протокол №5 від 25 травня 2018.*

### **Рисований О.М.**

Р54 Системне програмування Графічний інтерфейс користувача (GUI) : навчальний посібник для студентів спеціальностей 123 – «Комп'ютерна інженерія», 125 – «Кібербезпека» / О.М. Рисований – Харків : НТУ «ХПІ», 2018 – 160 с.

Розглянуто широке коло питань, починаючи з призначення віконного програмування, основ створення вікна в операційній системі Windows, атрибутів кольору та фону, перерисовування, відомостей про ресурси. Крім того, основну увагу приділено розгляданню інтерфейсів GDI та GDI+. Теоретичний матеріал підкріплено великою кількістю прикладів програмного коду – від самих коротких програм до програм середньої складності, виконаних в макроасемблері masm32. Для закріплення теоретичного матеріалу наведено лабораторні роботи з прикладами їх виконання.

Призначено для студентів спеціальностей 123 – «Комп'ютерна інженерія», 125 – «Кібербезпека»

Лл. 20. Бібліогр. 16 назв.

© О.М. Рисований, 2018 р.

## ЗМІСТ

<b>ВСТУП .....</b>	<b>5</b>
<b>1. ОСНОВИ СТВОРЕННЯ ВІКНА ОС WINDOWS .....</b>	<b>7</b>
1.1. Вікно та його створення .....	7
1.2. Повідомлення та їх оброблення .....	16
1.3. Перерисовування вікна .....	21
1.4. Атрибути кольору та фону вікна .....	26
1.5. Загальні відомості про ресурси .....	26
1.5.1. Меню .....	27
1.5.2. Ресурси у Visual Studio .....	31
1.6. Функції підтримки вікон .....	33
1.7. Смуги прокручування .....	35
1.8. Анімація вікна за допомогою функції AnimateWindow .....	40
1.9. Лабораторна робота “Windows-застосунок” .....	46
<b>2. ІНТЕРФЕЙС GDI. ГРАФІЧНІ МОЖЛИВОСТІ ТА ШРИФТИ ОС WINDOWS .....</b>	<b>51</b>
2.1. Обробка повідомлень WM_PAINT .....	52
2.2. Контекст пристрою .....	53
2.3. Графічні примітиви .....	54
2.3.1. Лінії та криві .....	55
2.3.2. Замкнуті фігури .....	57
2.4. Перо .....	60
2.4.1. Стандартне перо .....	61
2.4.2. Просте перо .....	62
2.4.3. Розширене перо .....	64
2.4.4. Косметичне перо .....	65
2.4.5. Геометричне перо .....	66
2.5. Щітки .....	67
2.5.1. Стандартні щітки .....	67
2.5.2. Щітка користувача .....	68
2.5.3. Суцільні щітки .....	68
2.5.4. Штрихові щітки .....	68
2.5.5. Растрові щітки .....	69
2.5.6. Структура LOGBRUSH і функція CreateBrushIndirect .....	70
2.6. Зміна характеристик графічних інструментів .....	71
2.7. Створення логічних шрифтів .....	81
2.8. Виведення тексту .....	83
2.9. Спіраль Архімеда з плавною зміною кольору фігури .....	92

2.10. Створення двох вікон на прикладі фігури у вигляді рози .....	101
2.11. Лабораторна робота “Графічні примітиви” .....	112
2.12. Лабораторна робота “Графічні фігури” .....	123
<b>3. ІНТЕРФЕЙС GDI+ .....</b>	<b>127</b>
3.1. Інтерфейс керованих класів .....	127
3.2. Методи рисування за допомогою графічних об’єктів .....	128
3.3. Ініціалізація та завершення .....	128
3.4. Клас Graphics .....	129
3.5. Клас GraphicsPath .....	135
3.6. Клас Region .....	139
3.7. Власні елементи управління .....	141
3.8. Створення геометричних фігур .....	142
3.9. Виведення тексту .....	153
<b>Список літератури .....</b>	<b>158</b>

## ВСТУП

Графічний інтерфейс користувача (англ. Graphical User Interface, GUI) – система взаємодії користувача із програмним застосунком, яка оснований на поданні всіх доступних користувачеві системних об'єктів і функцій у вигляді графічних компонентів екрана (вікон, меню, панелей інструментів, кнопок, елементів діалогу). На відміну від інтерфейсу командного рядка користувач має вільний доступ (за допомогою клавіатури або іншого пристрою введення) до всіх видимих екранних об'єктів.

При програмуванні під операційну систему Windows необхідно навчитися використовувати інтерфейс цієї системи. А інтерфейс системи можна запрограмувати за допомогою функцій Win API [1, 2, 7-11]. У навчальному посібнику розглянуто приклади з використанням Win32 API.

У 64-розрядній Windows функції Win64 API зазнали незначних змін. Назви деяких з них були змінені так, щоб відобразити належність до 64-розрядної платформи. Win64 допускає використання і 32-, і 64-розрядних даних. Продуктивність 32-розрядних застосунків при роботі на 64-розрядній платформі знижується. Це пов'язано з тим, що для запуску 32-розрядного застосунку операційній системі доводиться виконувати ряд підготовчих дій: збільшувати всі 32-розрядні показники до розміру у 8 байт, перетворювати виклики API-функцій, замінювати типи 32-розрядних даних на 64-розрядних [12, 13].

Win64-код об'єднує в собі основні можливості 32-розрядного коду, а також включає зміни, пов'язані з підвищенням розрядності. У розподіленні програміста виявляються:

- 64- розрядні показники;
- 64- розрядні типи даних;
- 32- розрядні типи даних;
- інтерфейс Win64 API.

Якою б не була платформа під Windows, без програмування інтерфейсу практично не обійтися.

Для проведення інформаційної розвіддільності потрібні знання, аналогічні тим, що можуть застосовуватися під час війни в кіберпросторі.

У цій книзі описується розробка застосунків з використанням інтерфейсу прикладного програмування (Application Programming Interface, API) операційних систем Windows компанії Microsoft. Windows API є найважливішим чинником, який впливає на весь процес розробки застосунків.

Інтерфейс Windows передбачає підмножину функцій інтерфейсу прикладного програмування Win API. Проте відштовхуватися тільки від документації Microsoft не рекомендується, оскільки в ній описується абстрактний інтерфейс Win API.

Навчальний посібник містить необхідні теоретичні відомості, практичні рекомендації і завдання з виконання робіт. Розглянуто обробку повідомлень від клавіатури, мишки та таймера, питання створення процесу, управління пам'яттю, реєстром, основи перехоплення API-функцій у WINDOWS.

У навчальному посібнику наведено безліч текстів програм, які можуть бути корисними для практичного застосування.

# 1. ОСНОВИ СТВОРЕННЯ ВІКНА ОС WINDOWS

## 1.1. Вікно та його створення

З погляду користувача, **вікно** – це прямокутна ділянка екрана, що відповідає застосуванню або його частині. Застосування може керувати декількома вікнами, серед яких виділяють одне головне вікно-рамку (Frame Window).

З погляду операційної системи, **вікно** – це в більшості випадків кінцевий пункт, до якого прямують повідомлення.

З погляду програміста, **вікно** – це об'єкт, атрибути якого (тип, розмір, положення на екрані, вигляд курсора, меню, значок, заголовок) повинні бути спочатку сформовані, а потім зареєстровані системою. Маніпуляція з вікном здійснюється за допомогою спеціальної віконної функції, яка має свою структуру.

Етапи створення вікна без оброблення повідомлень операційною системою та віконної процедури для обробки повідомлень вікном [4, 9-13]:

1. Створення образу вікна – класу, який визначає ціле сімейство вікон процедурою `WndClassEx` (`WndClassEX`).
2. Реєстрація класу процедурою `RegisterClassEX`, щоб образ вікна став доступний в програмі.
3. Створення вікна процедурою `CreateWindowEX` за зразком, який задано класом (<http://msdn.microsoft.com/ru-ru/library/>).

При віконному програмуванні різноманітні імена повідомлень мають префікс, який позначає категорію, до якої належить відповідна константа (табл. 1.1).

Таблиця 1.1 – Префікси для числових констант

Префікс	Категорія
CS_	Опція стилю класу
CW_	Опція створення вікна
DT_	Опція рисування тексту
IDC_	Ідентифікатор зумовленого курсора
IDI_	Ідентифікатор зумовленої ікони (піктограми)
WM_	Повідомлення вікна
WS_	Стиль вікна

ОС Windows є багатозадачною системою і може бути запущена кілька разів. Для того щоб розрізнити екземпляри програм, кожному екземпляру присвоюється **умовний номер – хендл (handle) або дескриптор**, який є указівником на блок пам'яті, в якому розмішений той або інший об'єкт. Дескриптор у Win32 присвоюється як самому вікну, так і всім елементам вікна: меню, курсору, іконі і так далі. Згідно з угорською нотацією ідентифікатори змінних типу HANDLE повинні починатися з букви h і знаходяться в пам'яті. Це означає, що в тих випадках, коли необхідно отримати хендл того або іншого об'єкта, то необхідно отримати адресу завантаженого в пам'ять об'єкта!

За **перший етап** (створення образу вікна) відповідає структура WNDCLASSEX, яка обробляє всі повідомлення і має структуру:

```

WNDCLASSEX STRUCT ; функція створення образу вікна
    cbSize    DWORD ? ; розмір структури
    style     DWORD ? ; стиль вікна
    lpfnWndProc  DWORD ? ; адреса процедури, яка реагує на повідомлення
    cbClsExtra  DWORD ? ; кількість байтів для структури
    cbWndExtra  DWORD ? ; кількість байтів для додаткових структур
    hInstance  DWORD ? ; дескриптор програми створення вікна
    hIcon      DWORD ? ; дескриптор «великої» піктограми
    hCursor    DWORD ? ; визначає курсор
    hbrBackGround  DWORD ? ; описує колір заповнення вікна
    lpszMenuName  DWORD ? ; визначає ім'я ресурсу меню
    lpszClassName  DWORD ? ; визначає ім'я класу
    hIconSm    DWORD ? ; маленьке віконце
WNDCLASSEX ENDS

```

Параметри цієї структури такі:

- **cbSize** – число байтів, зайнятих структурою WNDCLASSEX, що зазвичай визначається як **sizeof WNDCLASSEX**;
- **style** – визначає поведінку вікна даного класу. Поле style задається константами, що починаються з букв CS\_. Всі вони дорівнюють степеням двійки, тому їх об'єднання оператором OR рівнозначне установці відповідних «прапорців». Наприклад, **CSVREDRAW** дорівнює  $2^0$ , а **CS\_HREDRAW** –  $2^1$ . Їх об'єднання **CS\_HREDRAW** or **CS\_VREDRAW** установить нульовий і перший двійкові розряди поля style, а це означає, що вікно даного класу повинне перерисуватись при зміні як горизонтального, так і вертикального розмірів (табл. 1.2).



Таблиця 1.2 – Параметри структури WNDCLASSEX

Стиль	Опис
CS_VREDRAW	Перерисувати все вікно, якщо змінений розмір за вертикаллю
CS_HREDRAW	Перерисувати все вікно, якщо змінений розмір за горизонталлю
CS_KEYCVTWINDOW CS_DBLCLKS	Посилати повідомлення віконної функції при подвійному клацанні мишею, якщо курсор знаходиться в межах вікна
CS_OWNDC	Виділити унікальний контекст пристрою для кожного вікна, створеного за допомогою цього класу
CS_CLASSDC	Один і той же контекст пристрою розділяється всіма вікнами цього класу
CS_PARENTDC	Дочірні вікна успадковують контекст батьківського вікна
CS_NOKEYCVT CS_NOCLOSE	Забронити команду Close в системному меню
CS_SAVEBITS	Зберігати частину ділянки екрана, закриту вікном, як bitmap, при видаленні відновлювати перекриту ділянку
CS_BYTEALIGNCLTENT	Вирівнює межу робочої області вікна (у горизонтальному напрямі) так, щоб для відображення рядка було потрібне ціле число байтів
CS_BYTEALIGN-WINDOW	Те ж, але дію зачіпає все вікно
CS_GLOBALCLASS	Створити клас, доступний всім застосункам. Зазвичай цей стиль застосовується для створення визначуваних користувачем елементів управління в DLL

- lpfnWndProc – адреса процедури, яка реагує на повідомлення, що прийшли від вікна;
- cbClsExtra, cbWndExtra – ці поля використовуються рідко і тому дорівнюють нулю;
- hInstance – дескриптор програми, в якій створюється вікно;
- hIcon – дескриптор «великої» піктограми, яка показується на екрані при натисненні клавіш Alt+tab;

- **hCursor** – дескриптор курсора миші. Звичайна стрілка задається константою **IDC\_ARROW**. Але якщо задати курсор як **IDC\_WAIT**, то курсор миші перетвориться на зображення пісочного годинника;
- **hbrBackGround** – описує колір заповнення вікна. Найчастіше цей колір білий або сірий;
- **lpszMenuName** – адреса імені меню. Якщо меню не використовується, то параметр дорівнює **NULL**;
- **lpszClassName** – указівник на рядок з нульовим символом у кінці, який містить ім'я класу вікна. Це ж ім'я має бути у параметрі **lpszClassName** функції **CreateWindow()**.
- **hIconSm** – дескриптор «маленької» піктограми, яка показується на панелі завдань **Windows** у лівому верхньому кутку вікна. Розміри піктограми повинні бути 16x16 пікселів. Якщо поле дорівнює **NULL**, то система шукає ресурс піктограми, вказаний полем **hIcon**, щоб сформувати малу піктограму з нього.

Поле **hIcon** містить дескриптор піктограми, яка призначена для цього класу вікна. *Піктограма* – це маленька бітова картинка, яка з'являється на панелі завдань **Windows** і в лівій частині заголовка вікна. Значення **hIcon** зазвичай отримують викликом функції **LoadIcon**, яка має такий прототип:

**LoadIcon(HINSTANCE hInstance, LPCTSTR lpszIconName)**

Ця функція завантажує ресурс піктограми, заданий параметром **lpszIconName** з екземпляра застосування, вказаного параметром **hInstance**. Функцію можна використовувати також для завантаження однієї з системних піктограм, якщо передати першому аргументу значення **NULL**. В цьому випадку другий аргумент повинен містити константу, ідентифікатор якої починається з префікса **IDI\_** («ідентифікатор значка» – **ID for icon**). Можливі значення другого аргументу для зумовлених піктограм наведено в табл. 1.3.

Таблиця 1.3 – Зумовлені ідентифікатори піктограм

Значення	Опис
<b>IDI_APPLICATION</b>	Піктограма застосування за умовчанням
<b>IDI_ASTERISK</b>	Те саме, що й <b>IDI_INFORMATION</b>
<b>IDI_ERROR</b>	Піктограма у вигляді білого хреста на фоні червоного кола. Вона використовується в серйозних застережливих повідомленнях

Закінчення табл. 1.3

IDI_EXCLAMATION	Те саме, що й IDI_WARNING
IDI_HAND	Те саме, що й IDI_ERROR
IDI_INFORMATION	Піктограма «i», яка використовується в інформаційних повідомленнях
IDI_QUESTION	Піктограма «?»
IDI_WARNING	Піктограма «!», яка використовується в застережливих повідомленнях
IDI_WINLOGO	Логотип Windows

Поле `hCursor` містить дескриптор курсора миші, використовуваного застосунком в клієнтській ділянці вікна. Значення `hCursor` зазвичай отримують за допомогою виклику функції `LoadCursor`, що має такий прототип:

```
LoadCursor(HINSTANCE hInstance. LPCTSTR lpcursorname)
```

Ця функція завантажує ресурс курсора, заданий другим параметром (`lpCursorName`), з екземпляра застосунку, заданого першим параметром (`hInstance`).

Функцію можна також використовувати для завантаження одного з системних (зумовлених) курсорів, якщо передати першому аргументу значення `NULL`. Значення другого аргументу при цьому вибирається з табл. 1.4.

Таблиця 1.4 – Зумовлені ідентифікатори курсора

Значення	Опис
IDC_ARROW	Стандартна стрілка
IDC_APPSTARTING	Стандартна стрілка і малий пісочний годинник
IDC_CROSS	Перехрестя
IDC_HELP	Стрілка і знак питання
IDC_IBEAM	Текстовий двутавр
IDC_NO	Перекреслений кружок
IDC_SIZEALL	Чотирикінцева стрілка
IDC_SIZENESW	Двокінцева стрілка, яка вказує на північний схід і південний захід
IDC_SIZENS	Двокінцева стрілка, яка вказує на північ і південь

Закінчення табл. 1.4

IDC_SIZEWNSE	Двокінцева стрілка, яка вказує на північний захід і південний схід
IDC_SIZEWE	Двокінцева стрілка, яка вказує на захід і схід
IDC_UPARROW	Вертикальна стрілка
IDC_WAIT	Пісочний годинник

За **другий етап** створення вікна відповідає функція (процедура) RegisterClassEx, яка створений клас вікна робить доступним програмі, щоб та змогла на його образ і подобу створювати справжні вікна.

За **третій етап** створення вікна відповідають дві функції: CreateWindowEx – для створення вікна та ShowWindow – для його виведення на екран. Функція CreateWindowEx має такий вигляд:

```
invoke CreateWindowEx, \ ; створення вікна з розширеним стилем
NULL, \ ; додатковий стиль
ADDR ClassName, \ ; адреса імені класу
ADDR AppName, \ ; адреса імені вікна
WS_OVERLAPPEDWINDOW, \ ; базовий стиль
CW_USEDEFAULT, \ ; горизонтальна координата вікна
CW_USEDEFAULT, \ ; вертикальна координата вікна
CW_USEDEFAULT, \ ; ширина вікна
CW_USEDEFAULT, \ ; висота вікна
NULL, \ ; дескриптор батьківського вікна
NULL, \ ; дескриптор меню
hInstance, NULL ; дескриптор програми
```

Параметр CW\_USEDEFAULT задає значення за умовчанням, а параметр WS\_OVERLAPPEDWINDOW – найпоширеніший стиль вікна. Список різноманітних стилів вікна наведено в табл. 1.5.

Таблиця 1.5 – Список різноманітних стилів вікна

Стиль	Опис
WS_OVERLAPPED	Вікно має заголовок і обрамляючу рамку
WS_TABSTOP	Вікно може отримувати клавіатурний фокус при натисненні користувачем кнопки Tab
WS_MAXIMIZEBOX	Біля вікна є кнопка максимізації
WS_GROUP	Вікно є першим вікном групи

Продовження табл. 1.5

WS_MINIMIZEBOX	Біля вікна є кнопка мінімізації
WS_THICKFRAME	Біля вікна є достатньо товста рамка, що дозволяє вікну змінювати розміри; використовується в основному для вікон діалогу. Заголовок біля вікна немає
WS_SYSMENU	Біля вікна є системне меню
WS_HSCROLL	Біля вікна є горизонтальна лінійка прокручування
WS_VSCROLL	Біля вікна є вертикальна лінійка прокручування
WS_DLGFRA	Біля вікна є рамка, яка зазвичай буває в діалогових вікнах
WS_BORDER	Біля вікна є тонка обмежувальна рамка
WS_CAPTION	Додавання рамки та заголовка: WS_BORDER   WS_DLGFRA
WS_MAXIMIZE	Створюється вікно, що спочатку було максимізоване
WS_CLIPCHILDREN	При прорисованні батьківського вікна ділянка, займана дочірніми вікнами, не прорисовується
WS_CLIPBLINGS	Дочірнє вікно, що перекриває інше дочірнє вікно, при прорисованні ділянки не прорисовується
WS_DISABLED	Створюється спочатку заборонене (неактивне) вікно
WS_VISIBLE	Створюється вікно, що спочатку відображається
WS_MINIMIZE	Створюється спочатку мінімізоване вікно
WS_CHILD	Створюється дочірнє вікно, що має за умовчанням тільки робочу ділянку, меню вікна цього стилю не мають
WS_POPUP	Створюється висхідне (popup) вікно
WS_TILED	WS_OVERLAPPED
WS_ICONIC	WS_MINIMIZED
WS_SIZEBOX	WS_THICKFRAME
WS_TILEDWINDOW	WS_OVERLAPPEDWINDOW

Закінчення табл. 1.5

WS_OVERLAPPEDWINDOW	WS_OVERLAPPED   WS_CAPTION   WS_SYSMENU   WS_THICKFRAME   WS_MINIMIZEBOX   WS_MAXIMIZEBOX
WS_POPUPWINDOW	WS_SYSMENU   WS_BORDER   WS_POPUP
WS_CHILDWINDOW	WS_CHTLD
WS_EX_ACCEPTFILES	Створюване вікно може приймати і передавати повідомлення
WS_EX_DLGMODALFRAME	Створюється вікно з подвійною рамкою, в якому може знаходитися додатковий заголовок. Використовувати тільки для значення dwEx-Style
WS_EX_NOPARENTNOTIFY	Створене дочірнє вікно не посилає батьківському вікну повідомлення WM_PARENTNOTIFY при створенні або руйнуванні дочірнього вікна
WS_EX_TOPMOST	Створене вікно повинне розташовуватися поверх усіх вікон, що не мають такого стилю, і залишатися вище за останні навіть в неактивному стані
WS_EX_TRANSPARENT	Створене вікно повинне бути прозорим
WS_EX_CLIENTEDGE	Компонент виглядає тривимірним

Наведені стилі не завжди сумісні один з одним. У табл. 1.6 наведено сумісність стилів. У ній символом "+" відмічені стилі, які можна використовувати для створення вікон, що перекриваються, тимчасових і дочірніх.

Таблиця 1.6 – Сумісність стилів

Ім'я константи	Вікно, що перекривається	Тимчасове вікно	Дочірнє вікно
WS_BORDER	+	+	+
WS_CAPTION	+	+	+
WS_CHILD			+
WS_CHILDWINDOW			+
WS_CLIPCHILDREN	+	+	+
WS_CLIPSIBLINGS			+
WS_DISABLED	+	+	+

Продовження табл. 1.6.

WS_DLGFRAE	+	+	+
WS_GROUP			+
WS_HSCROLL	+	+	+
WS_ICONIC	+		
WS_MAXIMIZE	+		
WS_MAXIMIZEBOX	+	+	+
WS_MINIMIZE	+		
WS_MINIMIZEBOX	+	+	+
WS_OVERLAPPED	+		
WS_OVERLAPPEDWINDOW	+		
WS_POPUP		+	
WS_POPUPWINDOW		+	
WS_SYSMENU	+	+	+
WS_TABSTOP			+
WS_THICKFRAME	+	+	+
WS_VISIBLE	+	+	
WS_VSCROLL	+	+	+
WS_TILED	+		
WS_SIZEBOX	+	+	+
WS_TILEDWINDOW	+		
MDIS_ALLCHILDSTYLES			

Функція ShowWindow відображає вікно на екрані. У цій функції перший параметр – це дескриптор вікна, а другий – вигляд вікна, основні значення якого наведені в табл. 1.7.

Таблиця 1.7 – Основні значення функції ShowWindow

Значення	Опис
SW_HIDE	Вікно сховане
SW_SHOWNORMAL, SW_NORMAL	Вікно показане в його нормальних розмірах
SW_SHOWMINIMIZED	Вікно згорнуте та показане як піктограма
SW_SHOWMAXIMIZED, SW_MAXIMIZE	Вікно розгорнуте

Закінчення табл. 1.7

SW_SHOWNOACTIVE	Вікно відображається в його розмірах і позиції, установлених безпосередньо перед поточними значеннями розмірів і позиції. Активне вікно залишається активним
SW_SHOW	Вікно відображається в його поточних розмірах і позиції
SW_MINIMIZE	Вікно згорнуте й активізує вікно верхнього рівня в списку системи
SW_SHOWMINNOACTIVE	Вікно згорнуте. Активне вікно залишається активним
SW_SHOWNA	Вікно показане в його поточному стані. Активне вікно залишається активним
SW_RESTORE	Активізувати і відобразити вікно. Якщо вікно згорнуте або розгорнуте, йому будуть повернуті його розміри і позиція
SW_SHOWDEFAULT	Застосовується при запуску застосунку за замовчуванням

## 1.2. Повідомлення та їх оброблення

Операційна система WINDOWS спроектована як система для паралельного програмування, в якій використана ідея клієнт-серверної взаємодії. Процес, при якому виконується певна дія, називається сервером, а процес, для якого виконується дія, називається клієнтом. Клієнт і сервер обмінюються повідомленнями, або мають спільні блоки пам'яті. Ядро операційної системи складається з множини файлів типу \*.DLL (Dynamic Link Libraries), які є постачальниками певних послуг для клієнтів. Ці послуги доступні клієнтам через так звані функції API (Application Programming Interface).

Серед функцій API, які найчастіше використовуються, є функція SendMessageA, що надсилає повідомлення в чергу будь-якого процесу. Як правило, процеси отримують повідомлення з черги за допомогою функції GetMessageA.

Віконна процедура є сервером вікна. Вона виконує всі дії, які замовляє процес-клієнт за допомогою повідомлень, кожне з яких запускає віконну процедуру з відповідними параметрами за допомогою функції Dispatch-Message.



Старше вікно називають батьківським (parent), а молодше – дочірнім (child). Спеціалізовані вікна називають контролами (Controls). Вони мають свої стандартизовані назви (класи) і є дочірніми вікнами деякого батьківського вікна, яке відповідає за введення будь-яких параметрів. Контролами можна керувати так само, як і звичайними вікнами – за допомогою керуючих повідомлень.

Існують також інші функції відправлення повідомлень, які призначені для драйверів (SendDriverMessage), та функції відправлення повідомлень до паралельних ниток-процесів (PostThreadMessageA).

Програма може отримати повідомлення за допомогою API-функцій GetMessageA та PeekMessageA, які при отриманні повідомлення заповнюють шаблон цієї структури. Надіслати повідомлення можна будь-якою з таких функцій:

- SendMessageA – відправити повідомлення і чекати відповіді;
- PostMessageA – відправити повідомлення в чергу і йти далі;
- SendMessageCallbackA – отримати негайну (позачергову) відповідь на повідомлення та результат передати Callback-функції, яка паралельно з основною програмою буде опрацьовувати результат відповіді;
- SendMessageTimeoutA – відправити повідомлення вікна і якщо відповіді немає, то чекати певний час;
- BroadcastSystemMessage – відправити повідомлення в чергу до певних процесів або драйверів, чекаючи або не чекаючи відповіді.

При надсиланні повідомлення система сформує структуру MSG, яку можна буде отримати в іншому процесі. Перші чотири значення структури MSG формуються програмістом за допомогою функції SendMessageA, а три наступних параметри автоматично формуються операційною системою. Для того щоб прийняти повідомлення, необхідно оголосити структуру MSG в програмі і виконати одну з таких функцій:

- GetMessageA – чекати повідомлення і отримати результат в EAX;
- PeekMessageA – перевірити, чи є повідомлення в черзі.

У Windows 3.1 елементи управління посилали повідомлення про свій стан, наприклад, натисненням кнопки або зміною вибору, використовуючи повідомлення WM\_COMMAND з кодом стану, наприклад, BN\_CLICKED і ідентифікатором, поміщеним в lParam, і дескриптором в wParam.

### **Універсальне повідомлення WM\_NOTIFY**

У WIN32 підтримка старих повідомлень залишилася, але тепер за необхідності передачі нових даних вирішено використовувати не створення нових повідомлень, оскільки це пов'язано з переробкою API, а

універсальне повідомлення WM\_NOTIFY. Разом з ним можна передати будь-яку кількість додаткових даних.

Повідомлення WM\_NOTIFY відправляється стандартним органом управління своєму батьківському вікну, коли відбулася подія або орган управління вимагає деякої інформації. Щоб відправити це повідомлення, треба викликати функцію SendMessage.

Кожен новий елемент управління Windows підтримує такі сповіщення (<http://www.grinchuk.lviv.ua/books/78.html>):

NM\_CLICK – натиснення на кнопку;

NM\_DBLCLK – подвійне натиснення на кнопку;

NM\_RCLICK – натиснення правою кнопкою;

NM\_RDBLCLK – подвійне натиснення правою кнопкою;

NM\_RETURN – натиснення ENTER і елемент управління має фокус;

NM\_SETFOCUS – отримання фокусу;

NM\_KILLFOCUS – втрата фокусу;

NM\_OUTOFMEMORY – невістачання пам'яті.

Якщо обробник повідомлення знаходиться в процедурі діалогового вікна, то використовується функція SetWindowLong з прапорцем DWL\_MSGRESULT, щоб установити значення, яке повертається.

Будь-яке вікно призначене для прийому повідомлень. **Повідомлення** – це структура даних, яка містить елементи, такі, як дескриптор вікна, якому адресоване повідомлення; код (тип) повідомлення та додаткову інформацію. Повна інформація про повідомлення в ОС Windows міститься в структурі MSG:

MSG STRUCT

hwnd        DWORD ? ; дескриптор джерела повідомлення

message    DWORD ? ; тип повідомлення

wParam     DWORD ? ; параметр повідомлення

lParam     DWORD ? ; параметр повідомлення

time        DWORD ? ; час приходу повідомлення

pt POINT <> ; структура для зберігання координат MSG

MSG ENDS

де

- hwnd – дескриптор джерела повідомлення;
- message – тип повідомлення (наприклад WM\_LBUTTONDOWN – повідомлення про те, що піднята ліва кнопка миші);
- wParam та lParam – параметри повідомлення, що залежать від його типу;
- time – час приходу повідомлення;
- pt – структура, що зберігає координати повідомлення.

У табл. 1.8 наведено найбільш часто використовувані повідомлення.

Таблиця 1.8 – Найбільш часто використовувані повідомлення

Повідомлення	Призначення
WM_CLOSE	Повідомляє вікно про те, що воно повинне бути закрите. Повідомлення може бути використано для виведення запиту користувачеві з пропозицією підтвердити завершення роботи, перш ніж буде викликана функція DestroyWindow. За умовчанням (якщо оброблення WM_CLOSE не передбачено) DefWindowProc викликає DestroyWindow
WM_COMMAND	Повідомлення посилається, коли користувач вибирає команду меню або посилає команду з елемента управління
WM_CREATE	Посилається, коли застосунок створює вікно викликом функції CreateWindow або CreateWindowEx. Віконна процедура отримує це повідомлення, коли вікно вже створене, але ще не показано на екрані. Тому, обробляючи це повідомлення, можна змінити характеристики вікна або виконати деякі дії, наприклад, відкрити необхідні файли. Після оброблення повідомлення застосунок повинен повернути нульове значення для продовження процесу створення вікна. Якщо застосунок поверне значення -1, то вікно не буде створено, а функція CreateWindow поверне значення NULL
WM_DESTROY	Посилається віконній процедурі знищеного вікна після того, як вікно видалено з екрана
WM_INITDIALOG	Повідомлення посилається віконній процедурі діалогового вікна безпосередньо перед тим, як воно буде відображено на екрані. Оброблення цього повідомлення дозволяє провести ініціалізацію тих об'єктів, які пов'язані з діалоговим вікном
WM_MOVE	Посилається вікну, яке перемістилося на екрані
WM_PAINT	Посилається вікну, вміст якого вимагає перерисовування
WM_SIZE	Посилається вікну, розміри якого змінилися

Закінчення табл. 1.8

WM_TIMER	Повідомляє вікно про те, що деякий системний таймер, установлений функцією SetTimer, відлічив заданий йому інтервал
----------	---------------------------------------------------------------------------------------------------------------------

Повідомлення WM\_CLOSE з'являється тоді, коли натискають на кнопку перехрестя або на кнопки Alt + F4. Наявність цього повідомлення не обов'язкова, тому що коли воне відсутнє, функція DefWindowProc викличе за умовчанням функцію знищення вікна DestroyWindow. Її доцільно застосовувати для виведення додаткових застережливих повідомлень.

Довга *черга повідомлень* розбивається ОС на декілька дрібних. Кожна така черга обробляється конкретною програмою в циклі:

```
.WHILE TRUE ; поки істинне, то
invoke GetMessage, ADDR msg, NULL, 0, 0 ; прийом повідомлення
or eax, eax ; формування ознак
jz Quit ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправлення на обслуговування
до WndProc proc
.ENDW ; закінчення циклу оброблення повідомлень
Quit:
```

Функція GetMessage приймає повідомлення, призначене для даної програми, а DispatchMessage відправляє його функції WndProc, яка обслуговує конкретне вікно. У простому випадку така функція реагує тільки на повідомлення WM\_DESTROY, яке свідчить про те, що вікно, яке його послало, в даний момент знищується (наприклад, через натиснення "хрестика" на вікні):

```
WndProc proc hWnd:HWND, uMsg:UINT, \
wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_DESTROY ; якщо тип "дорівнює знищенню"
invoke PostQuitMessage,NULL ; то передача WM_QUIT
.ELSE ; інакше
invoke DefWindowProc, hWnd, uMsg, wParam, lParam ; оброблення та
; відправлення повідомлення до функції WndProc
ret ; повернення з процедури WndProc
.ENDIF
```

**Віконна процедура** – це функція зворотного виклику, яка призначена для оброблення повідомлень, що адресовані вікну того віконного класу, в якому міститься посилання на цю процедуру. Функцією зворотного виклику називають таку функцію, яку викликає сама операційна

система. З цієї причини в кодї програми немає прямого виклику віконної процедури. Функція `WndProc` обробляє повідомлення, послані їй `Windows`. Тільки `Windows` може викликати процедуру `WndProc()`.

Параметри процедури `WndProc`:

- `hWnd` – дескриптор примірника вікна, що пересилає повідомлення;
- `uMsg` – фактичний ідентифікатор (тип) повідомлення;
- `wParam` та `lParam` – інформаційні 16-розрядні параметри для повідомлення (старше та молодше слова) [5–8].

Функція `WndProc` визначає існування вікна. Якщо воно існує, то повідомлення `WM_DESTROY` не виникає, і всякі інші повідомлення відправляються стандартній функції `DefWindowProc`, у якій вони і обробляються. Якщо ж вікно знищується, функція `PostQuitMessage` генерує повідомлення `WM_QUIT`, яке стає в загальну чергу, а потім передає управління функції `GetMessage`.

Функція `GetMessage` відповідає на повідомлення `WM_QUIT` тим, що повертає нуль в регістр `eax`. Тому цикл `WHILE` припиняється і програма завершує роботу, переходячи до мітки `QUIT`.

Функція `DefWindowProc()` передбачає оброблення за умовчанням повідомлень, не оброблених у застосунку.

Параметри:

- `hWnd` – дескриптор вікна, що отримує повідомлення;
- `uMsg` – поле найменування повідомлення;
- `wParam` та `lParam` – параметри повідомлення.

### 1.3. Перерисовування вікна

Для того щоб вивести щось на екран, необхідно перерисувати вікно. Для перерисовування вікна служить повідомлення `WM_PAINT`. Це повідомлення посилає функція `UpdateWindow`.

Повідомлення спрямовується безпосередньо віконній процедурі, минаючи чергу повідомлень. Якщо ділянка оновлення порожня, повідомлення не відправляється.

Для того щоб програма могла відобразити на екрані повідомлення, необхідно додати вітку `ELSEIF`.

Таким чином, загальний текст віконної програми з оброблення повідомлень та віконною процедурою складається з 5-ти етапів:

1. Створення образу вікна.
2. Реєстрація класу.
3. Створення вікна.

4. Оброблення повідомлень.

5. Віконна процедура.

Стартовою функцією застосунку є функція **WinMain proc**, яка в загальному вигляді має 4 параметри:

**WinMain PROC PUBLIC hInstance, hPrevInstance, lpCmdLine, nCmdShow**

Параметр **hInstance** – дескриптор поточного екземпляра даного застосунку. Це значення використовується надалі для взаємодії з елементами робочого середовища, пов'язаними з даним застосунком: ресурсами, вікнами, зображеннями і багатьма іншими. Цей дескриптор локальний в рамках даного екземпляра застосунку.

Параметр **hPrevInstance** – дескриптор попереднього екземпляра даного застосунку. У win32 цей параметр не використовується і завжди дорівнює 0.

Параметр **lpCmdLine** – вказівник на командний рядок.

Параметр **nCmdShow** – параметр, що описує, в якому вигляді рекомендується створити головне вікно застосунку: приховане, згорнуте, нормальне, розгорнуте і так далі. Для самостійного застосунку цей параметр можна проігнорувати.

Повний текст програми наведено в програмі 1.1.

### Програма 1.1:

```
.686 ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС
option casemap:none ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
uselib kernel32, user32, gdi32
WinMain proto hInst:HINSTANCE, CmdShow:DWORD
.data ; директива визначення даних
    ClassName db "Первый класс",0
    AppName db "Перше вікно",0
    Hello db "Перше повідомлення!!!",0
.data? ; директива невизначених даних
    hInstance HINSTANCE ?
.code ; директива початку сегмента команд
start: ; мітка початку програми з ім'ям start
    invoke GetModuleHandle, NULL ; отримання дескриптора програми
    mov hInstance,eax ; збереження дескриптора програми
    invoke WinMain, hInstance,SW_SHOWDEFAULT
    invoke ExitProcess,eax ; повернення управління та вивільнення ресурсів
WinMain proc hInst:HINSTANCE, CmdShow:DWORD
LOCAL wc:WNDCLASSEX ; резервування стека під структуру
LOCAL msg:MSG ; резервування стека під структуру MSG
```

```

LOCAL hwnd:HWND ; резервування стека під хендл програми
push hInstance ; збереження в стеку дескриптора програми
pop wc.hInstance ; повернення дескриптора в поле структури
mov wc.cbSize,SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
mov wc.lpszWndProc, OFFSET WndProc ; адреса процедури WndProc
mov wc.hbrBackground,COLOR_WINDOW+1 ; колір вікна
invoke GetStockObject, WHITE_BRUSH ; читання описувача
mov wc.hbrBackground, eax ; колір заповнення вікна
mov wc.lpszMenuName,NULL ; ім'я ресурсу меню
mov wc.lpszClassName,OFFSET ClassName ; ім'я класу
invoke LoadIcon,NULL,IDI_APPLICATION ; відображення піктограми
mov wc.hIcon,eax ; дескриптор «великої» піктограми
mov wc.hIconSm,eax ; дескриптор маленького віконця
invoke LoadCursor,NULL,IDC_ARROW ; відображення курсора
mov wc.hCursor,eax
mov wc.cbClsExtra,NULL ; кількість байтів для структури класу
mov wc.cbWndExtra,NULL ; кількість байтів для структури вікна
invoke RegisterClassEx, ADDR wc ; функція реєстрації класу вікна
invoke CreateWindowEx, \ ; функція створення вікна за зразком
NULL,ADDR ClassName, \ ; стиль та адреса імені класу
ADDR AppName,\ ; адреса імені вікна
WS_OVERLAPPEDWINDOW, \; базовий стиль вікна
CW_USEDEFAULT,CW_USEDEFAULT,\; гориз. та верт. координати вікна
CW_USEDEFAULT,CW_USEDEFAULT, \ ; ширина та висота вікна
0,0,hInst,0 ; дескриптори батьківського вікна, меню, програми
mov hwnd,eax
invoke ShowWindow,hwnd,SW_SHOWNORMAL ; видимість вікна

.WHILE TRUE ; поки істинне, то
invoke GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення
or eax, eax ; формування ознак
jz Quit ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправлення до WndProc proc
.ENDW ; закінчення циклу оброблення повідомлень
Quit:
mov eax,msg.wParam
ret ; повернення з процедури WinMain

WinMain endp ; закінчення процедури з ім'ям WinMain

WndProc proc hwnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM

```

```

LOCAL hdc:HDC ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під PAINTSTRUCT

```

```

LOCAL rect:RECT ; резервування стека під структуру RECT

.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
invoke PostQuitMessage,NULL; передача повідомлення WM_Quit

.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури та
; заповнення структури
mov hdc,eax ; збереження контексту
invoke GetClientRect,hWnd, ADDR rect ; занесення в структуру rect
; характеристик вікна
invoke DrawText,hdc,ADDR Hello,-1, ADDR rect, \ ; рисування тексту
DT_SINGLELINE or DT_CENTER or DT_VCENTER
invoke EndPaint,hWnd, ADDR ps ; закінчення рисування

.ELSE ; інакше
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; оброблення та
; відправлення повідомлення до функції WndProc
ret

.ENDIF ; закінчення логічної структури .IF – .ELSEIF
xor eax,eax
ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

Вигляд вікна, створеного відповідно до програми 1.1, наведено на рис. 1.1.

Директиви LOCAL повинні йти безпосередньо після директиви PROC. Директива LOCAL має параметри <ім'я\_змінної>:<тип\_змінної>.

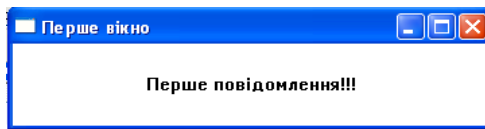


Рис. 1.1. Вигляд вікна, створеного відповідно до програми з лістингу 1.1

Директива LOCAL ws:WNDCLASSEX повідомляє асемблера MASM'у, що слід зарезервувати пам'ять із стека в обсязі, що дорівнює розміру структури WNDCLASSEX для змінної розміром ws. Це дозволяє звернутися до ws без маніпуляції із стеком. Але локальні змінні не можуть бути використані поза функцією, в якій вони були створені та будуть автоматично знищені функцією після повернення управління .



Для отримання виконуваного файлу з ім'ям, наприклад 21\_1l.exe, необхідно використовувати командний файл

```
ml /c /coff "21_1L.asm"  
link /SUBSYSTEM:windows "21_1L.obj"
```

Для завершення роботи застосування необхідне одне з повідомлень: WM\_DESTROY чи WM\_QUIT.

При виборі пункту виходу застосування з вікна (або кнопки закриття) виконується:

1. Видалення вікна за допомогою функції DestroyWindow().
2. Виклик повідомлення WM\_DESTROY.
3. Оброблення WM\_DESTROY.
4. Виконання функції PostQuitMessage().
5. Виклик повідомлення WM\_QUIT.

Процедура BeginPaint отримує загальні параметри пристрою, але їй не відоме положення вікна і його розміри.

Процедура GetClientRect заносить у спеціальну структуру rect положення вікна і його розміри.

Рисування вікна відбувається за процедурою DrawText, яка керує п'ятьма параметрами:

- дескриптором контексту;
- початковою адресою рядка символів;
- числом символів (якщо рядок завершується нулем – ставиться -1);
- адресою структури rect, що описує положення і розмір вікна;
- параметром, що визначає стиль виведення. Цей параметр можна задати за допомогою бітових прапорців, сполучених операторами OR. У нашому випадку за горизонталлю (DT\_CENTER) і вертикаллю (DT\_VCENTER).

Після виведення рядка контекст пристрою вже не потрібний, і щоб не захащувати пам'ять зайвими параметрами, його слід видалити процедурою EndPaint.

Моргання тексту вікна відбувається тому, що відповідний клас вікна створювався з параметрами CS\_HREDRAW or CS\_VREDRAW, які задають перерисовування вікна при зміні його горизонтальних (CS\_HREDRAW) або вертикальних (CS\_VREDRAW) розмірів. Як тільки вікно змінюється, Windows передає повідомлення WM\_PAINT – і напис рисується наново.

## 1.4. Атрибути кольору та фону вікна

Для встановлення кольору тексту в клієнтській ділянці вікна може використовуватися функція `SetTextColor`. Для встановлення кольору фону графічного елемента, який відображається позаду кожного символу, використовується функція `SetBkColor`. Для регулювання режиму змішування фону (*background mix mode*) викликається функція `SetBkMode`. Її другий параметр може набувати значення `OPAQUE`, якщо колір фону графічного елемента виводиться поверх існуючого фону вікна, перш ніж буде виведений символ. Також можна використовувати значення `TRANSPARENT`, при якому колір фону графічного елемента ігнорується, а символ виводиться на існуючому фоні.

Якщо ці функції не викликалися, то в контексті пристрою будуть використані значення за умовчанням, які наведено в табл. 1.9.

Таблиця 1.9 – Атрибути кольору і фону тексту за умовчанням

Атрибут	Значення за умовчанням
Колір тексту	Чорний
Колір фону графічного елемента	Білий
Режим змішування фону	OPAQUE

## 1.5. Загальні відомості про ресурси

Ресурси – це зумовлені дані системи або дані програми користувача, які визначаються ще до початку роботи програми, особливим чином додаються у виконуваний файл і використовуються при роботі програми.

У Windows є деякі зумовлені дані, такі, як курсори, ікони та шітки. Окрім них, до ресурсів належать:

- використовувані в програмі зображення;
- рядки символів;
- меню;
- прискорювачі клавіатури;
- діалогові вікна;
- шрифти;
- ресурси, визначувані користувачем.

Використання ресурсів дає дві цілком певні вигоди:

– ресурси завантажуються в пам'ять лише при зверненні до них, тобто реалізується економія пам'яті;

– властивості ресурсів підтримуються системою автоматично, не вимагаючи від програміста написання додаткового коду.

До ресурсів, які визначаються користувачем, належать і програми .exe, і файли динамічної бібліотеки .dll, а також інші бінарні файли. Всі вони називаються **bin-файлами**.

Ресурси визначаються до початку роботи програми та додаються в **bin-файл**. При завантаженні **bin-файлу** в пам'ять ресурси в пам'ять не завантажуються.

Ресурси створюються окремо від файлів програми та додаються в **bin-файл** при лінкованні програми.

Опис ресурсів зберігається окремо від програми в текстовому файлі (\*.rc) і компілюється (\*.res) спеціальним транслятором ресурсів. У виконуваний файл ресурси включаються компонувальником. Транслятором ресурсів у пакеті MASM32 є RC.EXE.

Такі ресурси, як ікони, курсори, діалогові вікна, зображення (bitmap), зберігаються в окремих файлах з розширеннями .ico, .cur, .dig, .bmp відповідно. У rc-файлах робляться посилання на ці файли.

Файл ресурсів компілюється спеціальним компілятором ресурсів rc.exe. Після компіляції файлу ресурсів компілятором ресурсів створюється новий файл, що має розширення .res. Цей res-файл використовується лінкером для додавання ресурсів у **bin-файл**.

Найпростіший шлях створення ресурсів – це використання інтегрованих середовищ з **редактором ресурсів** (ResED, Visual Studio та ін.). Вони дозволяють створювати ресурси, візуально контролювати правильність їх створення, а потім зберігати їх у файлах ресурсів.

### ***1.5.1. Меню***

Існує два види меню:

- головне меню вікна;
- висхідне (контекстне) меню.

Основним елементом меню, що відображається, є **рядок** або **графічний об'єкт**.

Будь-яке вікно може мати меню. **Головне меню** знаходиться нижче від заголовка вікна, його рядки розташовані в одну або декілька ліній. При виборі рядка головного меню, як правило, активізується розділ меню. **Розділом меню** є тимчасове меню.

Рядки **тимчасового меню** розташовані в один або декілька стовпців. Якщо тимчасове меню може з'являтися в будь-якому місці робочої ділянки, то воно називається плаваючим. У деяких випадках зручніше застосовувати **плаваюче меню**.

За способом створення розрізняють **статичне і динамічне меню**. Статичне меню створюють до запуску і не змінюють у процесі роботи

застосування. Динамічне меню створюють у процесі роботи застосунку. Динамічне меню після створення можна змінювати або залишати незмінним. Робота з рядками статичних і динамічних меню нічим не відрізняється.

Меню можна створити декількома способами:

1. Створення меню безпосередньо в програмі (без файлу опису ресурсів) за допомогою функцій `CreateMenu`, `AppendMenu` та `InsertMenu`;

2. Створення меню безпосередньо в програмі (без файлу опису ресурсів) за допомогою функції `LoadMenuIndirect`, створивши заздалегідь структуру `MENUITEMPLATE`;

3. Визначення меню у файлі опису ресурсів:

– з описом ресурсів на мовах високого рівня;

– з використанням редакторів ресурсів (`ResED`, `Visual Studio` та ін.).

Самий простий шлях – це задання меню в спеціальному файлі з розширенням `.rc`. Розширення `.rc` – це файл з ресурсами (рисунки, опис меню, курсори тощо).

У випадку, якщо необхідно додати функціональну кнопку меню, при натисканні на яку не з'являються висхідні кнопки, а виконується дія, програма ресурсів може бути такою:

```
#define IDM_AVTOR 4
#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_ABOUT 3
FirstMenu MENU
{
    MENUITEM "Автор",IDM_AVTOR
    POPUP "Программа"
    {
        MENUITEM "Функции",IDM_HELLO
        MENUITEM SEPARATOR
        MENUITEM "Окончание",IDM_EXIT
    }
    POPUP "Результаты"
    {
        MENUITEM "About",IDM_ABOUT
    }
}
```

У наведеному файлі ресурсів такою функціональною кнопкою є кнопка **"Автор"**, при натисканні на яку оброблюється повідомлення `IDM_AVTOR`.

**Програма 1.2.** Файл `menu.rc` задання меню:

```

#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_ABOUT 3
FirstMenu MENU {
    POPUP "програма"{
        MENUITEM "Функции",IDM_HELLO
        MENUITEM SEPARATOR
        MENUITEM "Окончание",IDM_EXIT
    }
    POPUP "результаты" {
        MENUITEM "About",IDM_ABOUT
    }
}

```

Це меню має два пункти: "програма" та "результаты". Пункт "програма" має два підпункти "Функции" та "Окончание", які розділені за допомогою слів MENUITEM SEPARATOR рисою.

У головній програмі (програма 1.3) потрібно задати константи, які відповідають пунктам меню.

При запуску створеного ехе-файлу відображається меню, яке наведено на рис. 1.2.

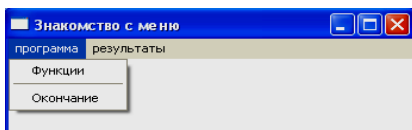


Рис. 1.2. Меню програми при запуску ехе- файла

### Програма 1.3. Головна програма управління меню:

```

.686 ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\include\user32.inc ; файли інтерфейсу ...
include \masm32\include\kernel32.inc ; файли функцій застосувань...
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
IDM_HELLO equ 1
IDM_EXIT equ 2 ; вихід із програми
IDM_ABOUT equ 3

.data ; директива визначення даних
ClassName db "SimpleWinClass",0
AppName db "Знакомство с меню",0
MenuName db "FirstMenu",0

```

```

Hello_string db "Ну, здравствуй!",0
Goodbye_string db "Прощай!",0
About_string db "Испытание меню",0
wc WNDCLASSEX <>
msg MSG <>
hwnd HWND ?
hInstance HINSTANCE ?

```

```

.code          ; директива початку сегмента команд
start:        ; мітка початку програми з ім'ям start

```

```

invoke GetModuleHandle, NULL ; отримання дескриптора програми
mov hInstance,eax ; збереження дескриптора програми
mov wc.cbSize,SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
mov wc.lpfnWndProc, OFFSET WndProc ; адреса процедури WndProc
mov wc.cbClsExtra,NULL ; кількість байтів для структури
mov wc.cbWndExtra,NULL ; кількість байтів для додаткових структур
push hInstance ; перезапис через стек
pop wc.hInstance ; формування поля дескриптора
mov wc.hbrBackground,COLOR_WINDOW+1 ; колір вікна
mov wc.lpszMenuName,OFFSET MenuName ; ім'я ресурсу меню
mov wc.lpszClassName,OFFSET ClassName ; ім'я класу
invoke LoadIcon,NULL,IDI_APPLICATION ; ресурс піктограми
mov wc.hIcon,eax ; дескриптор «великої» піктограми
mov wc.hIconSm,eax ; дескриптор маленького віконця
invoke LoadCursor,NULL,IDC_ARROW ; ресурс курсора
mov wc.hCursor,eax
invoke RegisterClassEx, ADDR wc ; функція реєстрації класу вікна
invoke CreateWindowEx, 0,ADDR ClassName, \ ; стиль та адреса імені класу
ADDR AppName, WS_OVERLAPPEDWINDOW, \ ; базовий стиль
CW_USEDEFAULT,CW_USEDEFAULT,\ ; гориз. та верт. координати вікна
CW_USEDEFAULT,CW_USEDEFAULT, \ ; ширина та висота вікна
NULL,0, hInst,0 ; дескриптори батьківського вікна, меню, програми
mov hwnd,eax
invoke ShowWindow, hwnd,SW_SHOWNORMAL ; видимість вікна
.WHILE TRUE ; поки істинне, то
invoke GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення
or eax, eax ; формування ознак
jz Quit ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправка до WndProc proc
.ENDW
Quit:
mov eax,msg.wParam

```

```
invoke ExitProcess, eax
```

```
WndProc   proc   hWnd:HWND,   uMsg:UINT,   wParam:WPARAM,  
IParam:LPARAM
```

```
.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна  
    invoke PostQuitMessage,NULL ; передача повідомлення
```

```
.ELSEIF uMsg==WM_COMMAND ; якщо є повідомлення від меню  
    mov eax,wParam
```

```
.IF ax==IDM_HELLO ; якщо є повідомлення IDM_HELLO
```

```
    invoke MessageBox, 0,ADDR Hello_string, OFFSET AppName,MB_OK
```

```
.ELSEIF ax==IDM_ABOUT ; якщо є повідомлення IDM_ABOUT
```

```
    invoke MessageBox,0,ADDR About_string,OFFSET AppName, MB_OK
```

```
.ELSE ; інакше
```

```
    invoke DestroyWindow,hWnd ; знищення вікна
```

```
.ENDIF
```

```
.ELSE
```

```
invoke DefWindowProc,hWnd,uMsg,wParam,IParam ; обробка повідомлень
```

```
ret
```

```
.ENDIF
```

```
xor eax,eax ; підготування до завершення
```

```
ret ; повернення управління ОС
```

```
WndProc endp ; закінчення процедури
```

```
end start ; закінчення програми з ім'ям start
```

Командний **bat**-файл для створення .exe-файлу повинен бути та-  
ким:

```
ml /c /coff "21_2L.asm"
```

```
rc "menu.rc"
```

```
link /SUBSYSTEM:windows "21_2L.obj" "menu.res"
```

```
pause
```

```
start 21_2L.exe
```

### *1.5.2. Ресурси в Visual Studio*

Для створення файлу ресурсів у середовищі Visual Studio необхідно запусити середовище.

Для створення проекту можна вибрати шлях: File/New/Project... або у відкритому вікні з ім'ям Start Page вибрати пункт Project у рядку Create:. У вікні, яке з'явилося, необхідно вибрати пункт MFC Application, ввести ім'я свого проекту та натиснути <OK> (рис. 1.3).

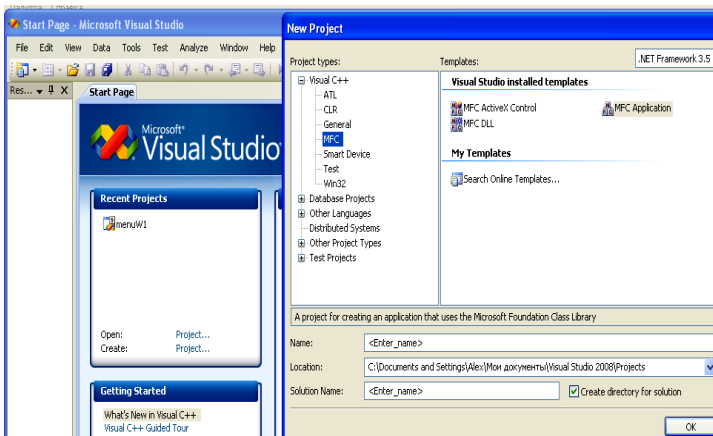


Рис. 1.3. Вікно створення проекту

Після виконаних дій з'явиться нове вікно з ім'ям MFC Application Wizard (майстер застосунків). Переконайтеся, що відзначений пункт Multiple documents. У лівій частині вікна з ім'ям MFC Application Wizard треба вибрати пункт Application Type та натиснути <Finish>.

Після цього в лівому вікні з назвою Solution Explorer середовища відобразяться всі створені файли (рис. 1.4).

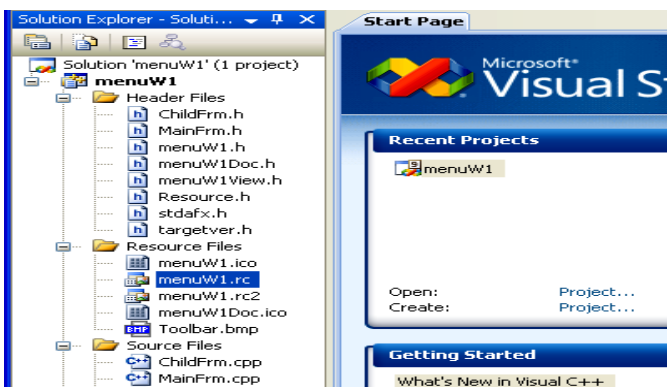


Рис. 1.4. Вікно Solution Explorer зі створеними файлами

Вибираємо ім'я створеного файлу ресурсів та натискаємо двічі



ліву клавішу мишки. Вікно згортається та відображаються всі компоненти ресурсів. При натисканні на пункт Menu відображаються два типи складових меню (рис. 1.5).

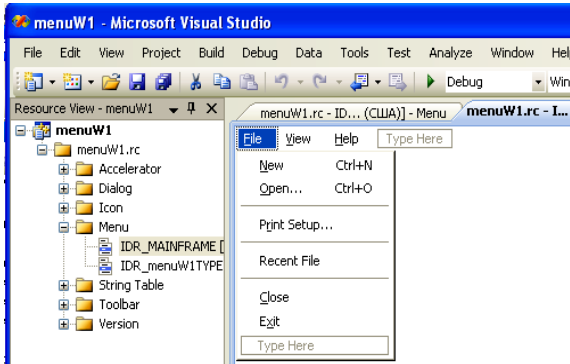


Рис. 1.5. Вікно з пунктами меню

Лівою клавішею мишки необхідно підсвітити небажані елементи меню, натиснути на праву клавішу мишки та видалити їх.

Зберегти введені зміни через меню вікна File/Save. Закрити середовище Visual Studio 2008, після чого знайти файл ресурсів через файловий менеджер та видалити небажані рядки програми.

## 1.6. Функції підтримки вікон

Win32 API надає широкий набір функцій, які дозволяють змінювати розміри, розташування і характеристики відображення вікна, основні з яких наведені в табл. 1.10.

Таблиця 1.10 – Основні API-функції, які змінюють розміри вікна

Функція	Призначення
BringWindowToTop	Активізує вікно і переносить його у верхнє положення, якщо воно знаходиться позаду інших вікон
CloseWindow	Згортає вікно
EnableWindow	Дозволяє або забороняє вікну приймати введення даних від миші або клавіатури
FindWindow	Знаходить вікно верхнього рівня за заданими іменем класу й іменем вікна
GetActiveWindow	Вибирає дескриптор активного вікна

Закінчення табл. 1.10

GetClassInfo	Витягує інформацію про клас вікна із структури типу WNDCLASS
GetClassLong	Вибирає специфіковане за індексом значення із структури типу WNDCLASSEX
GetClientRect	Отримує координати клієнтської ділянки вікна
GetFocus	Вибирає дескриптор вікна, що має фокус введення
GetParent	Вибирає дескриптор батьківського вікна
GetWindow	Вибирає дескриптор вікна
GetWindowInfo	Витягує інформацію про вікно із структури типу WINDOWINFO
GetWindowLong	Вибирає вказане за індексом значення з даних вікна
GetWindowRect	Отримує координати лівого верхнього і правого нижнього кутів вікна
IsChild	Визначає, чи є вікно дочірнім відносно даного батьківського вікна
IsIconic	Перевіряє, чи згорнуте вікно
IsZoomed	Перевіряє, чи розгорнуте вікно
IsWindowEnabled	Перевіряє, чи дозволене або заборонене вікно
MoveWindow	Переміщає і змінює розміри вікна
SetActiveWindow	Робить вікно активним
SetClassLong	Заміщає вказаним 32-бітовим значенням колишнє значення вказаного поля в структурі WNDCLASSEX для класу вікна, до якого належить вказане вікно
SetLayered-WindowAttributes	Установлює світлопроникність та прозорість забарвлення клавіші багатошарового вікна
SetFocus	Додає вікну фокус введення
SetWindowLong	Змінює атрибути вказаного вікна (спочатку задані функцією CreateWindow)
SetWindowPos	Змінює одночасно розміри, позицію і розташування вікна щодо інших вікон
SetWindowText	Функція встановлює текст усередині елемента управління або текст заголовка вікна.
ShowWindow	Відображає, приховує або змінює стан показу вікна
UpdateWindow	Оновлює клієнтську ділянку вікна за допомогою посилання повідомлення WM_PAINT безпосередньо віконній процедурі, минувши чергу повідомлень застосунку

Наприклад, фрагмент програми для згортання і розгортання панелі завдань на робочому столі монітора 9 разів:

```
...
first db 'Shell_TrayWnd', 0
.code
start:
mov ebx, 1
.while (ebx < 10)
invoke FindWindow, addr first, 0 ; витягує дескриптор вікна верхнього рівня
invoke ShowWindow, eax, SW_SHOW ; показ вікна
invoke Sleep, 200
invoke FindWindow, addr first, 0
invoke ShowWindow, eax, SW_HIDE ; приховує вікно і активізує інше вікно
invoke Sleep, 200
inc ebx
.endw
invoke FindWindow, addr first, 0
invoke ShowWindow, eax, SW_SHOW
invoke ExitProcess, 0
end start
```

## 1.7. Смуги прокручування

Функція `CreateWindow` має два стилі – `WS_HSCROLL` та `WS_VSCROLL`. Ці стилі визначають наявність біля вікна горизонтальної та вертикальної смуг прокручування. Смуги прокручування є частиною вікна. Діапазон прокрутки визначає число кроків між крайніми позиціями бігунка (слайдера). За умовчанням для смуг прокручування цей діапазон визначений від 0 до 100. Для того щоб змінити діапазон прокручування, необхідно викликати функцію `SetScrollRange`, яка визначена таким чином:

```
SetScrollRange(HWND hWnd, int nBar, int nMinPos, int nMaxPos, BOOL bRedraw)
```

Перший аргумент функції – хендл вікна, якому належать смуги прокручування. Другий аргумент визначає, для якої смуги прокрутки (вертикальної або горизонтальної) встановлюється діапазон. Аргумент може набувати значення `SB_VERT` або `SB_HORZ`, що визначає роботу з вертикальною або горизонтальною смугою прокручування. Третій і четвертий аргументи безпосередньо вказують на нижню й верхню межу діапазону прокручування. П'ятим аргументом є прапорець, що визначає, чи потрібно перерисувати смугу прокручування після визначення

діапазону. Якщо значення **TRUE**, то смуга прокручування перерисовувалася, а якщо **FALSE** – перерисовування не потрібне. При діапазоні прокручування від 0 до 0 смуга прокручування стає невидимою.

Дії на смуги приводять до того, що функція вікна, якій належать смуги прокручування, отримує повідомлення **WM\_VSCROLL** (якщо дії проводилися вертикальною смугою) або **WM\_HSCROLL** (реакція на дію на горизонтальну смугу).

Молодше подвійне слово **wParam** визначає характер дії на смугу прокручування і може набувати значень, наведених у табл. 1.11.

Таблиця 1.11 – Значення молодшого подвійного слова **wParam**

Параметр	Опис
<b>SB_LINEUP</b>	Використовується тільки з <b>WM_VSCROLL</b> ; клацання мишею на стрілці вгору приводить до прокручування на один «рядок» вгору
<b>SB_LUNELEFT</b>	Використовується тільки з <b>WM_HSCROLL</b> ; клацання мишею на стрілці вліво приводить до прокручування на одну «колонку» вліво
<b>SB_LINEDOWN</b>	Використовується тільки з <b>WM_VSCROLL</b> ; клацання мишею на стрілці вниз приводить до скролінгу в одній «лінії» вниз
<b>SB_LINERIGHT</b>	Використовується тільки з <b>WM_HSCROLL</b> ; клацання мишею на стрілці вправо приводить до прокрутки на одну «колонку» вправо
<b>SB_PAGEUP</b>	Використовується тільки з <b>WM_VSCROLL</b> ; клацання мишею на смугі прокрутки вище слайдера приводить до прокрутки на одну «сторінку» вгору
<b>SB_PAGELEFT</b>	Використовується тільки з <b>WM_HSCROLL</b> ; клацання мишею на смугі прокрутки лівіше слайдера приводить до прокрутки на одну «сторінку» вліво
<b>SB_PAGE-DOWN</b>	Використовується тільки з <b>WM_VSCROLL</b> ; клацання мишею на смугі прокрутки нижче слайдера приводить до прокрутки на одну «сторінку» вниз
<b>SB_PAGE-RIGHT</b>	Використовується тільки з <b>WM_HSCROLL</b> ; клацання мишею на смугі прокрутки правіше слайдера приводить до прокрутки на одну «сторінку» управо
<b>SB_THUMB-POSITION</b>	Перетягування слайдера закінчене, користувач віджав клавішу миші
<b>SB_THUMBT-RACK</b>	Слайдер перетягується за допомогою миші, приводить до переміщення вмісту екрана

Продовження табл 1.11.

SB_TOP	Використовується тільки з вертикальними смугами прокручування, реалізованими як дочірні вікна; користувач натиснув клавішу «Home»
SB_LEFT	Використовується тільки з горизонтальними смугами прокручування, реалізованими як дочірні вікна; користувач натиснув клавішу «Home»
SB_BOTTOM	Використовується тільки з вертикальними смугами прокрутки, реалізованими як дочірні вікна; користувач натиснув клавішу «End»

Розглянемо програму створення вікна з пунктами меню та з горизонтальною смугою прокручування. Файл ресурсів наведено в програмі 1.4.

**Програма 1.4.** Файл ресурсів:

```
#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_ABOUT 3
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL "mk_icon.ico"
FirstMenu MENU {
    POPUP "Программа"{
        MENUITEM "Задание",IDM_HELLO
        MENUITEM SEPARATOR
        MENUITEM "Выйти",IDM_EXIT
    }
    POPUP "Информация"{
        MENUITEM "About",IDM_ABOUT
    }
}
```

У програмі з програми 1.5 використовуються дві піктограми: одна піктограма – стандартна з параметром `IDI_WARNING`, а друга – піктограма, яка відображається в панелі завдань екрана монітора. Горизонтальна смуга прокручування визначається в директиві створення вікна `invoke CreateWindowEx,NULL, ADDR ClassName,\ ADDR AppName, WS_OVERLAPPEDWINDOW or WS_HSCROLL,\ 0, 0,\ ; гориз. та верт. координати вікна 250,450,NULL,NULL,\ ; ширина та висота вікна hInstance,NULL`

**Програма 1.5:**

```
.686          ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
```

```
uselib user32, kernel32, gdi32
```

```
IDM_HELLO equ 1
IDM_EXIT  equ 2
IDM_ABOUT equ 3
IDI_ICON  equ 22
```

```
.data          ; директива визначення даних
ClassName db "SimpleWinClass",0
AppName   db "Окно с параметрами",0
MenuName  db "FirstMenu",0
Hello_string db "Создать окно с такими параметрами:",10,13
str1 db "- пиктограмма '!";",10,13
str2 db "- курсор в виде перечёркнутого круга;",10,13
str3 db "- у окна есть горизонтальная полоса прокрутки;",10,13
str4 db "- вид окна функции ShowWindow: окно отображается в его текущих размерах и позиции;",10,13
str5 db "- окно размещено в левом верхнем углу;",10,13
str6 db "- размеры окна: 250 на 450 точек.",0
About_string db "Выполнил: ",10,13
str7 db " e-mail: ",10,13,0
wc WNDCLASSEX <>
msg MSG <>
hwnd HWND ?
hInstance HINSTANCE ?
```

```
.code          ; директива початку сегмента команд
start:         ; мітка початку програми з ім'ям start
```

```
invoke GetModuleHandle, 0 ; отримання дескриптора програми
mov hInstance, eax ; збереження дескриптора програми
mov wc.cbSize, SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
push hInstance ; збереження в стеку дескриптора програми
pop wc.hInstance ; повернення дескриптора в поле структури
mov wc.hbrBackground, COLOR_WINDOW+1; колір вікна
mov wc.lpszMenuName, OFFSET MenuName ; ім'я ресурсу меню
mov wc.lpszClassName, OFFSET ClassName ; ім'я класу
```

```

invoke LoadIcon,NULL,IDI_WARNING ; піктограми у вигляді оклику
    mov wc.hIcon,eax ; дескриптор «великої» піктограми
    mov wc.hIconSm,eax ; дескриптор маленького віконця
invoke LoadCursor,NULL, IDC_NO ; курсор у вигляді перекресленого
кружка
    mov wc.hCursor,eax

```

```

invoke RegisterClassEx, addr wc ; функція реєстрації класу вікна

```

```

invoke CreateWindowEx,NULL, ADDR ClassName,\
    ADDR AppName, WS_OVERLAPPEDWINDOW or WS_HSCROLL,\
    0, 0, 250,450,NULL,0,hInstance,NULL
    mov hwnd,eax

```

```

invoke ShowWindow, hwnd,SW_SHOW ; видимість вікна
invoke SetScrollRange, hwnd,SB_HORZ,0,10,TRUE

```

```

.WHILE TRUE ; поки істинне, то
    invoke GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення
    or eax, eax ; формування ознак
    jz Quit ; перейти на мітку Quit, якщо eax = 0
    invoke DispatchMessage, ADDR msg ; відправлення до WndProc
.ENDW ; закінчення циклу оброблення повідомлень
Quit:
    mov eax,msg.wParam

```

```

invoke ExitProcess, eax ; повернення управління та визволення ресурсів

```

```

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM

```

```

.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
    invoke PostQuitMessage,0 ; передача повідомлення WM_Quit

```

```

.ELSEIF uMsg==WM_COMMAND; якщо є повідомлення від меню
    mov eax,wParam
    .IF ax==IDM_HELLO ; якщо є повідомлення IDM_HELLO
    invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
    .ELSEIF ax==IDM_ABOUT ; якщо є повідомлення IDM_ABOUT
    invoke MessageBox,NULL,ADDR About_string, OFFSET AppName,
    MB_OK
    .ELSE
        invoke DestroyWindow,hWnd ; знищення вікна
    .ENDIF

```

```

.ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; стандартне оброблення
        ; повідомлень, які явно не оброблюються

```

```

ret ; повернення з процедури
.ENDIF
xor eax,eax
ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми

```

Результати роботи програми наведені на рис. 1.6.

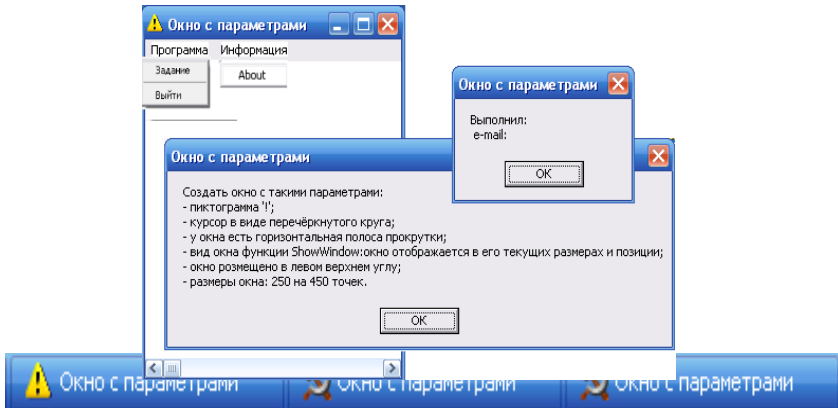


Рис. 1.6. Результаты работы програми

## 1.8. Анімація вікна за допомогою функції `AnimateWindow`

Щоб вікно вийшло як прозоре вікно, спочатку створюють звичайне вікно. Потім встановлюється багат шаровий біт розширеного стилю вікна `WS_EX_LAYERED` та викликається функція `SetLayeredWindowAttributes` з бажаним `alpha`-значенням.

Таким чином, послідовність дій після того, як створено вікно, повинна бути такою:

- викликаємо **`AnimateWindow`** для створення спецефектів;
- викликаємо **`GetWindowLongA`**, яка змінює атрибути вікна;
- викликаємо **`SetLayeredWindowAttributes`** з бажаним `alpha`-значенням.

Функція `AnimateWindow` дає можливість справляти спеціальні враження при показі або захованні вікон. Є чотири типи мультиплікації: ролик, слайд, згортання або розгортання і плавне альфа-перетікання.

Синтаксис:



AnimateWindow(hwnd: HWND, dwTime: DWord, dwFlags: DWord);

Параметри:

hwnd – дескриптор вікна, яке анімується. Потік повинен володіти цим вікном;

dwTime – встановлює, скільки необхідно часу, щоб відтворити мультиплікацію в мілісекундах. Як правило, відтворення мультиплікації займає 200 мілісекунд;

dwFlags – визначає тип мультиплікації. Цей параметр може складатися з декількох значень. За умовчанням ці прапорці дають бажаний результат при показі вікна. Щоб отримувати бажаний результат при захованні вікна, використовується прапорець AW\_HIDE і логічний оператор OR з відповідними прапорцями:

AW\_SLIDE – використовує слайдову анімацію. За умовчанням використовується анімаційний ролик (прокручування мультиплікації).

Цей прапорець ігнорується тоді, коли він користується з прапорцем AW\_CENTER;

AW\_ACTIVATE – активізує вікно (при показі вікна). Це значення не використовується спільно з AW\_HIDE;

AW\_BLEND – використовує ефект поступової зміни прозорості вікна.

Прапорець може бути використаний тільки в тому випадку, якщо параметр hwnd – вікно верхнього рівня;

AW\_HIDE – приховує вікно. За умовчанням вікно показується на екрані;

AW\_CENTER – розташування по центру і стиснення в точку, якщо використовується прапорець AW\_HIDE;

AW\_HOR\_POSITIVE – анімує вікно зліва направо. Цей прапорець може бути використаний з роликом або слайдом мультиплікації.

Він ігнорується, коли використовується з прапорцем AW\_CENTER або AW\_BLEND;

AW\_HOR\_NEGATIVE – анімує вікно справа наліво. Цей прапорець може бути використаний з роликом або слайдом мультиплікації.

Він ігнорується, коли використовується з прапорцем AW\_CENTER або AW\_BLEND. Наприклад:

```
invoke AnimateWindow,hWin,400,AW_ACTIVATE or  
AW_HOR_NEGATIVE
```

```
invoke SetFocus,hWin
```

AW\_VER\_POSITIVE – анімує вікно зверху вниз. Цей прапорець може бути використаний з роликом або слайдом мультиплікації. Він ігнорується, коли використовується з прапорцем AW\_CENTER або AW\_BLEND;

**AW\_VER\_NEGATIVE** – анімує вікно від низу до верху. Цей прапорець може бути використаний з роликом або слайдом мультиплікації. Він ігнорується, коли використовується з прапорцем **AW\_CENTER** або **AW\_BLEND**.

Для того щоб отримувати розширені дані про помилки, треба викликати функцію **GetLastError**.

При використанні слайда або прокручування мультиплікації необхідно задати напрям: **AW\_HOR\_POSITIVE**, **AW\_HOR\_NEGATIVE**, **AW\_VER\_POSITIVE** або **AW\_VER\_NEGATIVE**. Можна комбінувати прапорці **AW\_HOR\_POSITIVE** або **AW\_HOR\_NEGATIVE** з **AW\_VER\_POSITIVE** або **AW\_VER\_NEGATIVE** для того, щоб анімувати вікно за діагоналлю.

Виконні процедури для вікна і його дочірніх вікон повинні обробляти будь-яке повідомлення **WM\_PRINT** або **WM\_PRINTCLIENT**. Діалогові вікна, органи управління і стандартні органи управління обробляють **WM\_PRINTCLIENT**. Задана за умовчанням віконна процедура обробляє **WM\_PRINT**.

Функція **GetWindowLong** витягує інформацію про визначуване вікно (змінює атрибути вікна) та має синтаксис:

```
LONG GetWindowLong(  
    HWND hWnd,  
    int nIndex  
);
```

Параметр **nIndex**:

**GWL\_EXSTYLE** – витягує дані про розширений стиль вікна;

**GWL\_STYLE** – витягує дані про стиль вікна;

**GWL\_WNDPROC** – витягує адресу віконної процедури або дескриптор, що позначає адресу віконної процедури. Необхідно використовувати функцію **CallWindowProc**, щоб викликати віконну процедуру;

**GWL\_HINSTANCE** – витягує дескриптор екземпляра застосунку;

**GWL\_HWNDPARENT** – витягує дескриптор батьківського вікна, якщо воно існує;

**GWL\_ID** – витягує ідентифікатор вікна;

**GWL\_USERDATA** – витягує призначені для користувача дані, пов'язані з вікном. Його значення спочатку є нульовим.

**Програма 1.6.** Програма зі зміною прозорості вікна:

```
.686 ; директива визначення типу мікропроцесора  
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows  
option casemap:none ; відмінність малих та великих літер
```

```
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
uselib kernel32, user32;
WinMain proto :DWORD, :DWORD, :DWORD, :DWORD
```

```
.data?
    hInstance HINSTANCE ?
    CommandLine LPSTR ?
    Value dd ?
```

```
.data
    ClassName db "MASM Builder",0
    Caption db "Исследование функции AnimateWindow",0
```

```
.code
start:
    invoke GetModuleHandle, NULL ; отримання дескриптора програми
    mov    hInstance, eax ; збереження дескриптора програми
    invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess, eax
```

```
WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, \
                                                    CmdShow:DWORD
```

```
    LOCAL wc :WNDCLASSEX
    LOCAL msg :MSG
    LOCAL hWnd :HWND
```

```
    mov wc.cbSize, SIZEOF WNDCLASSEX ; кількість байтів структури
    mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль
    (CS_BYTEALIGNCLIENT)
    mov wc.lpfnWndProc, offset WndProc ; адреса процедури WndProc
    mov wc.cbClsExtra, NULL ; кількість байтів для структури
    mov wc.cbWndExtra, NULL ; кількість байтів для структури
    push hInst ; збереження в стеці дескриптора програми
    pop wc.hInstance ; повернення дескриптора в поле структури
    mov wc.hbrBackground, COLOR_BTNFACE+1 ; сірий колір вікна
    mov wc.lpszClassName, OFFSET ClassName ; ім'я класу
    invoke LoadIcon, NULL, IDI_APPLICATION ; без піктограми
    mov wc.hIcon, eax ; дескриптор піктограми
    mov wc.hIconSm, eax ; дескриптор маленького віконця
    invoke LoadCursor, NULL, IDC_ARROW ; ресурс курсора
    mov wc.hCursor, eax
```

```
    invoke RegisterClassEx, addr wc ; реєстрація класу вікна
```

```
    invoke CreateWindowEx, 0, ADDR ClassName, ADDR Caption, \
```

```

WS_OVERLAPPEDWINDOW,350,80,400,200,0,0,hInst,0
mov hWnd,eax
invoke ShowWindow,hWnd,SW_SHOWNORMAL
invoke UpdateWindow,hWnd

```

```

.WHILE TRUE
invoke GetMessage,ADDR msg,0,0,0 ; читання повідомлення
.BREAK .IF (leax)
invoke TranslateMessage,ADDR msg
invoke DispatchMessage,ADDR msg ; відправлення на обслуговування
.ENDW
mov eax,msg.wParam
ret ; повернення з процедури
WinMain endp ; закінчення процедури з ім'ям WinMain

```

### WndProc proc

```
hWnd:HWND,uMsg:UINT,wParam:WPARAM,IParam:LPARAM
```

```

.IF uMsg == WM_DESTROY
invoke PostQuitMessage,NULL

.ELSEIF uMsg == WM_CREATE ;
invoke AnimateWindow,hWnd,1000,AW_BLEND+AW_ACTIVATE
invoke GetWindowLongA,hWnd,GWL_EXSTYLE ; or WS_EX_LAYERED
or eax,WS_EX_LAYERED
invoke SetWindowLongA,hWnd,GWL_EXSTYLE,eax
mov Value,150 ; прозорість вікна: 0 - повністю прозоре и 255 - непрозоре
invoke SetLayeredWindowAttributes,\ ; встановлює прозорість
hWnd,Value,Value,LMA_COLORKEY + LMA_ALPHA
.ELSE
invoke DefWindowProc,hWnd,uMsg,wParam,IParam
ret
.ENDIF ; закінчення логічної структури
xor eax,eax
ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

Після того як програма відлагоджена, її можна всю або частково переписати з використанням макросів, особисто макросу @ для розміщення командних слів одним рядком. Як приклад фрагмент програми може бути таким:

```

...
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
CmdShow:DWORD
LOCAL wc :WNDCLASSEX

```

```

LOCAL msg :MSG
LOCAL hWnd :HWND
@m<mov wc.cbSize,SIZEOF WNDCLASSEX>,<mov
wc.style,CS_HREDRAW or CS_VREDRAW>,<mov wc.lpfWndProc,offset
WndProc>
@m<mov wc.cbClsExtra,0>,<mov wc.cbWndExtra,0>,<push hInst>,<pop
wc.hInstance>
@m<mov wc.hbrBackground, COLOR_BTNFACE+1>,<mov wc.lpszClassName,OFFSET ClassName>,<invoke LoadIcon,0,
IDI_APPLICATION>
@m<mov wc.hIcon,eax>,<mov wc.hIconSm,eax>,<invoke LoadCursor,0,
IDC_ARROW>,<mov wc.hCursor,eax >
invoke RegisterClassEx,addr wc ; реєстрація класу вікна
invoke CreateWindowEx,0,ADDR ClassName,ADDR Caption, \
WS_OVERLAPPEDWINDOW, 350,80,400,200,0,0,hInst,0
    mov hWnd,eax

invoke ShowWindow,hWnd,SW_SHOWNORMAL
invoke UpdateWindow,hWnd

.WHILE TRUE
invoke GetMessage,ADDR msg,0,0,0 ; читання повідомлення
.BREAK .IF (leax)
invoke TranslateMessage,ADDR msg
invoke DispatchMessage,ADDR msg ; відправлення на обслуговування
.ENDW
mov eax,msg.wParam
ret ; повернення з процедури
WinMain endp ; закінчення процедури з ім'ям WinMain

WndProc proc
hWnd:HWND,uMsg:UINT,wParam:WPARAM,IParam:LPARAM
.IF uMsg == WM_DESTROY
invoke PostQuitMessage,NULL
.ELSEIF uMsg == WM_CREATE ;
invoke AnimateWindow,hWnd,1000,AW_BLEND+AW_ACTIVATE
invoke GetWindowLongA,hWnd,GWL_EXSTYLE; or WS_EX_LAYERED
or eax,WS_EX_LAYERED
invoke SetWindowLongA,hWnd,GWL_EXSTYLE,eax
mov Value,150 ; прозорість вікна: 0 - повністю прозоре и 255 - непрозоре
invoke SetLayeredWindowAttributes,\ ; встановлює прозорість
hWnd,Value,Value,LMA_COLORKEY + LMA_ALPHA
@m<.ELSE>,<invoke DefWindowProc,hWnd,uMsg,wParam,IParam>, <ret>,<.ENDIF>
@m<xor eax,eax>,<ret>,<WndProc endp>,<end start>

```

## 1.9. Лабораторна робота “Windows-застосунок”

### Мета заняття:

- поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;
- набути практичних навичок у побудові базового застосунку під Win32 з дослідженням параметрів віконних процедур.

### Постановка задачі

Написати програму створення вікна Windows з файлом ресурсів. Вікно повинне містити заголовок, у якому відображається прізвище студента, номер його навчальної групи та його електронна адреса. Зміна розмірів вікна повинна супроводжуватись повідомленням. Параметри ж самого вікна повинні відповідати варіанту завдання. **Функцію MessageBox у програмі використати тільки один раз.**

### Методичні рекомендації

У випадку, якщо треба взяти ресурс піктограми з іншого exe-файлу, то можна скористатися, наприклад, програмою PE Resource Explorer (wasm.ru).

У налагоджувачі OllyDbg при натисканні правої кнопки мишки на програмі : Search for / Name Ctrl+N отримаємо повний список імен API-шних функцій, які використані в програмі.

### Варіанти завдань

#### 1. Параметри вікна:

- піктограма застосунку за умовчанням;
  - курсор у вигляді стандартної стрілки та малого пісочного годинника;
  - вікно має заголовок і рамку з обрамленням;
  - вигляд вікна функції ShowWindow: вікно сховане;
  - вікно розташувати у лівому верхньому куті екрана;
  - розміри вікна 300 × 500 точок.
- Використовувати кнопки віконного меню в такому порядку:
- перша кнопка – “Результат”. У ній – висхідна кнопка “Выход из программы”;
  - друга кнопка – “Справка”. У ній – висхідні кнопки “Автор программы”, “Требования к программе”, “Выход из программы”;
  - третя кнопка “Выход из программы”.
- Використати у функції AnimateWindow параметр AW\_SLIDE.

#### 2. Параметри вікна:

- піктограма у вигляді білого хреста на фоні червоного круга;
  - курсор у вигляді перехрестя;
  - біля вікна є кнопка максимізації;
  - вигляд вікна функції ShowWindow: вікно показане в його нормальних розмірах;
  - вікно розташувати у правому верхньому куті екрана;
  - розміри вікна 200 × 550 точок.
- Використовувати кнопки віконного меню в такому порядку:
- перша кнопка – “О программе”;
  - друга кнопка – “Автор”. У ній – висхідна кнопка “Об авторе”;
  - третя кнопка – “Выход из программы”.
- Використати у функції AnimateWindow параметр AW\_CENTER.

### 3. Параметри вікна:

- піктограма «!»;
  - курсор у вигляді стрілки та знака питання;
  - біля вікна є кнопка мінімізації;
  - вигляд вікна функції ShowWindow: вікно згорнуте і показане як піктограма;
  - вікно розташувати у лівому нижньому куті екрана;
  - розміри вікна: 500 × 600 точок.
- Використовувати кнопки віконного меню в такому порядку:
- перша кнопка – “Справка”. У ній – висхідні кнопки “О программе”, “Об авторе”;
  - друга кнопка – “Результат”;
  - третя кнопка – “Выход из программы”.
- Використати у функції AnimateWindow параметр AW\_HOR\_POSITIVE.

### 4. Параметри вікна:

- піктограма «?»;
  - курсор у вигляді текстового двотавру;
  - біля вікна є достатньо товста рамка;
  - вигляд вікна функції ShowWindow: вікно відображається в його розмірах і позиції, встановлених безпосередньо перед значеннями розмірів і позиції;
  - вікно розташувати по центру екрана;
  - розміри вікна 200 × 650 точок.
- Використовувати кнопки віконного меню у такому порядку:
- перша кнопка – “Справка”. У ній – висхідні кнопки “О программе”, “Требования к программе”, “Об авторе”;

- друга кнопка – “Результат”;
- третя кнопка – “Выход из программы”.

Використати у функції `AnimateWindow` параметр `AW_HOR_NEGATIVE`.

#### 5. Параметри вікна:

- піктограма «!»;
- курсор у вигляді перекресленого кола;
- біля вікна є горизонтальна лінійка прокручування.
- вигляд вікна функції `ShowWindow`: вікно відображається в його поточних розмірах і позиції;
- вікно розташувати у лівому верхньому куті екрана;
- розміри вікна  $250 \times 700$  точок.

Використовувати кнопки віконного меню в такому порядку:

- перша кнопка – “Справка”;
- друга кнопка – “Результат”. У ній – висхідні кнопки “Выход из программы”, “Результат”.

Використати у функції `AnimateWindow` параметр `AW_VER_POSITIVE`.

#### 6. Параметри вікна:

- піктограма з логотипом `Windows`;
- курсор у вигляді чотирикінцевої стрілки;
- біля вікна є вертикальна лінійка прокручування;
- вигляд вікна функції `ShowWindow`: вікно згорнуте й активізує вікно верхнього рівня в списку системи;
- вікно розташувати у правому верхньому куті екрана;
- розміри вікна  $300 \times 750$  точок.

Використовувати кнопки віконного меню в такому порядку:

- перша кнопка – “О программе”;
- друга кнопка – “Результат”. У ній – висхідні кнопки “Результат”, “Об авторе”, “Выход”.

Використати у функції `AnimateWindow` параметр `AW_VER_NEGATIVE`.

#### 7. Параметри вікна:

- піктограма у вигляді білого хреста на фоні червоного круга;
- курсор у вигляді двокінцевої стрілки, яка вказує на північний схід і південний захід;
- біля вікна є рамка, яка зазвичай буває у діалогових вікон;
- вигляд вікна функції `ShowWindow`: вікно згорнуте;



- вікно розташувати у лівому нижньому куті екрана;
- розміри вікна 300 × 800 точок.

Використовувати кнопки віконного меню в такому порядку:

- перша кнопка – “Сведения про автора”;
- друга кнопка – “О программе”. У ній – висхідні кнопки “Требования к программе”, “Результат”, “Выход”
- третя кнопка: назва “Выход”.

Використати у функції `AnimateWindow` параметр `AW_CENTER`.

#### 8. Параметри вікна:

- піктограма «i»;
- курсор у вигляді двокінцевої стрілки, яка вказує на захід і схід;
- біля вікна є тонка обмежувальна рамка;
- вигляд вікна функції `ShowWindow`: вікно сховане;
- вікно розташувати у правому нижньому куті екрана;
- розміри вікна 250 × 850 точок.

Використовувати кнопки віконного меню в такому порядку:

- перша кнопка – “Сведения об авторе”;
- друга кнопка – “О программе”. У ній – висхідні кнопки “Требования к программе”, “Результат”, “Выход”;
- третя кнопка: назва “Выход”.

Використати у функції `AnimateWindow` параметр `AW_HOR_POSITIVE`.

#### 9. Параметри вікна:

- піктограма «!»;
- курсор у вигляді вертикальної стрілки;
- створюється висхідне (popup) вікно;
- вигляд вікна функції `ShowWindow`: вікно показане в його поточковому стані;
- вікно розташувати по центру екрана;
- розміри вікна 500 × 900 точок.

Використовувати кнопки віконного меню в такому порядку:

- перша кнопка – “Справка”. У ній – висхідні кнопки “Требования к программе”, “Сведения про автора”;
- друга кнопка – “О программе”. У ній – висхідні кнопки “Требования к программе”, “Результат”;
- третя кнопка: назва “Выход”.

Використати у функції `AnimateWindow` параметр `AW_VER_NEGATIVE`.

#### 10. Параметри вікна:

- піктограма у вигляді білого хреста на фоні червоного кола;
- курсор у вигляді пісочного годинника;
- створюється спливаюче (popup) вікно;
- вигляд вікна функції ShowWindow: вікно сховане;
- вікно розташувати у правому нижньому куті екрана;
- розміри вікна 500 × 950 точок.

Використовувати кнопки віконного меню в наступному порядку:

– перша кнопка – “Результат”. У ній – висхідні кнопки “Результат”, “Выход из программы”;

– друга кнопка: назва “Справка”. У ній – висхідні кнопки “Автор программы”, “Требования к программе”, “Выход из программы”;

– третя кнопка – “Выход из программы”.

Використати у функції AnimateWindow параметр AW\_BLEND.

#### Контрольні запитання

1. У чому полягає алгоритм створення вікна?
2. Які елементи структури повідомлення залежать від його типу?
3. Опис яких програмних об'єктів необхідно включити у текст застосування для реєстрації класу вікон?
4. У чому полягає порядок виклику функцій віконної процедури?
5. Що відбувається з повідомленнями Windows, які не обробляються у функції вікна?
6. Яка функція є функцією зворотного виклику й в чому її суть?

## 2. ІНТЕРФЕЙС GDI. ГРАФІЧНІ МОЖЛИВОСТІ ТА ШРИФТИ ОС WINDOWS

Середовище ОС Windows є графічним, тому робота з графікою стає невід'ємною частиною програм в ОС Windows. Для розробки програм, що працюють з графікою в середовищі Windows, існують такі інструменти:

- бібліотека OpenGL;
- бібліотека DirectX;
- стандартні засоби Windows для роботи з графікою (GDI, GDI+);
- засоби для роботи з графікою в середовищах розробки.

Розглянемо стандартні засоби для роботи з графікою, які надаються операційними системами сімейства Windows (<http://msdn.microsoft.com/ru-ru/library/cwka53ef.aspx>).

До стандартних засобів Windows для роботи з графікою належить інтерфейс графічних пристроїв (Graphics Device Interface, GDI).

Інтерфейсом графічних пристроїв GDI є сукупність програмних засобів Windows (на основі функцій Win32 API), що організують виведення зображень на різні пристрої – екран, пристрої друку або у файли графічних об'єктів (текстових рядків, геометричних фігур, растрових зображень та ін.).

Окрім виведення зображень, в завдання інтерфейсу GDI входить спостереження за межами, в яких здійснюється це виведення. GDI відображає на екрані тільки ті частини об'єктів, які потрапляють усередину вікна застосування. Так зроблено для того, щоб застосунки не затирали один одного.

GDI бере участь в організації управління декількома застосунками, причому вікна верхнього рівня відображаються цілком, а від вікон нижніх рівнів видно тільки частини, що відступають. Перерисовування кожного вікна ведеться в межах видимих меж цього вікна.

Для виведення графічної інформації існує набір функцій, які можна розділити на декілька категорій:

- методи рисунку ліній: LineTo, MoveTo, Polyline, Arc, ArcTo, PolyBezier та ін.;
- методи рисунку замкнутих фігур: Ellipse, Rectangle, Polygon, Pie, Chord та ін.;
- методи виведення тексту: TextOut, DrawText та ін.;
- функції роботи з растровим зображенням: GetPixel, SetPixel, FloodFill, BitBlt та ін.

Існує окрема категорія функцій роботи з DC щодо перемикання режимів і установці параметрів виведення графічної інформації. Частина з них установлюється безпосередньо через певні функції (наприклад SetBkColor), частина – за допомогою спеціальних графічних об'єктів:

- перо (pen) – задає режим виведення ліній (колір, товщина, стиль);
- кисть (brush) – регулює режим зафарбовування фігур (колір, стиль);
- шрифт (font) – задає властивості шрифту, яким виводиться текст;
- палітра (palette) – задає набір використовуваних у DC кольорів;
- ділянка (region) – використовується для задання clipping regions – ділянок відсікання, поза якими виведення графіки блокується.

Робота з графічними об'єктами проводиться за допомогою їх дескрипторів (handles) – HDC, HPEN, HBRUSH, HFONT і так далі. Створення і видалення об'єктів проводиться за допомогою відповідних функцій – наприклад, об'єкт pen створюється за допомогою CreatePen, а знищується за допомогою DeleteObject. Режими, які задаються через графічні об'єкти, перемикаються за допомогою створення нових об'єктів і вказівки контексту (DC) використовувати їх для виведення графіки. Це робиться за допомогою функції SelectObject. При виборі нового об'єкта через SelectObject як значення, яке повертається, передається дескриптор об'єкта, що був у використанні в DC раніше.

З роботою графічного інтерфейсу тісно пов'язане оброблення повідомлення WM\_PAINT.

## 2.1. Оброблення повідомлень WM\_PAINT

При виведенні графіки у Windows, один раз нарисовавши вікно, при будь-яких діях з цим вікном його треба перерисувати. Наприклад, потрібно скрутити вікно або закрити його частину іншим вікном, і все, що було нарисоване, пропадає.

Річ у тому, що Windows не зберігає вміст клієнтської частини вікна. До клієнтської частини вікна належить усе, окрім заголовка вікна і елементів, що управляють (controls): меню, панелей інструментів (toolbar), кнопок і так далі. Застосунок сам повинен поклопотатися про те, щоб відрисувати свої дані в клієнтській ділянці. Windows лише посилає йому повідомлення, коли це потрібно зробити. Робиться це за допомогою посилення вікну повідомлення WM\_PAINT.

Повідомлення WM\_PAINT включаються практично в будь-який застосунок Windows, якщо в ньому хоч що-небудь рисується на екрані. **Загальне правило рисування** полягає в тому, що виведення у вікно застосунку будь-яких графічних об'єктів (текстових рядків, геометричних

фігур, окремих точок, растрових зображень) повинне виконуватися виключно в процедурі оброблення повідомлення WM\_PAINT. Тільки в цьому випадку графічний вміст вікна не втрачатиметься, якщо дане вікно затулятиметься вікнами інших застосунків.

Копіювання вмісту вікна на нові місця екрана забезпечують системні програми Windows.

Якщо в застосунку є лінійка меню, то при розгортанні пунктів меню вони перекривають частину вікна. Затулену раніше ділянку вікна треба перерисувати. Це завдання Windows також бере на себе.

При розгортанні всього зображення Windows вже не бере на себе завдання збереження і відновлення зображення у вікні, а замість цього посилає в застосунок повідомлення WM\_PAINT.

Оброблення повідомлення WM\_PAINT пов'язане з використанням найважливішого поля даних Windows, званого контекстом пристрою.

При обробленні WM\_PAINT повинна бути викликана функція BeginPaint. BeginPaint повертає дескриптор DC, який повинен бути використаний для перерисування клієнтської частини вікна. При обробленні WM\_PAINT дескриптор DC вікна повинен бути отриманий саме з використанням BeginPaint, а звільнений EndPaint – тоді як у всіх інших випадках відрисовки потрібно використовувати інші функції (наприклад GetDC/ReleaseDC).

## 2.2. Контекст пристрою

**Контекст пристрою** (в даному випадку – вікна) є ділянкою пам'яті, що закріплюється за робочою ділянкою вікна, в якій зберігаються поточні значення режимів, пов'язаних з рисунням, а також дескриптори інструментів рисування – щітки, пера, шрифти тощо. Контекст пристрою виступає в ролі з'єднувальної ланки між застосунком і драйвером пристрою та є структурою даних розміром приблизно 800 байт. Усі графічні функції GDI використовують контекст пристрою для визначення режиму рисування і характеристик використовуваних інструментів.

Контекст пристрою належить до системних ресурсів, кількість яких у системі обмежена. Тому спочатку треба отримати у системі необхідний ресурс, а потім, закінчивши роботу з ним, – повернути його системі.

Таким чином, для того щоб вивести у вікно деяке зображення, необхідно виконати дії, послідовність яких, по суті, визначає алгоритм оброблення повідомлення WM\_PAINT:

- отримати в системі контекст пристрою для даного вікна;

- змінити за необхідності режими рисунка або характеристики конкретних інструментів;
- сформулювати за допомогою графічних функцій GDI необхідне зображення;
- повернути програмі Windows зайнятий у неї контекст пристрою, привівши його заздалегідь у початковий стан.

Якщо необхідно рисувати на пристрої графічного виведення (екрані дисплея або принтері), то спочатку потрібно отримати *дескриптор контексту пристрою*. Повертаючи цей дескриптор після виклику відповідних функцій, Windows тим самим надає розробникові право на використання даного пристрою. Після цього дескриптор контексту пристрою передається як параметр у функції GDI, щоб ідентифікувати пристрій, на якому повинне виконуватися рисунка.

Контекст пристрою містить багато атрибутів і система витягує з контексту пристрою ті, які їй потрібні.

Win32 API підтримує такі типи контекстів пристрою:

- контекст дисплея;
- контекст принтера;
- контекст у пам'яті (сумісний контекст);
- метафайловий контекст;
- інформаційний контекст.

### 2.3. Графічні примітиви

Рисунка будь-якої фігури можна було б звести до багаторазового виклику функції `SetPixel` з відповідною зміною координат  $x$  та  $y$ . Однак подібний спосіб рисунка не зовсім зручний.

З кривими та лініями використовуються функції, наведені в табл. 2.1.

Таблиця 2.1 – Функції, які використовуються з кривими та лініями

Функція	Опис
<code>AngleArc</code>	Функція креслить відрізок прямої та дугу
<code>Arc</code>	Функція рисує еліптичну дугу
<code>ArcTo</code>	Функція рисує еліптичну дугу
<code>GetArcDirection</code>	Функція витягує поточний напрям дуги для вказаного контексту пристрою
<code>LineDDA</code>	Функція встановлює, які пікселі повинні виділятися для лінії заданими точками
<code>LineDDAProc</code>	Функція зворотного виклику, використовується з функцією <code>LineDDA</code>

Закінчення табл. 2.1

LineTo	Функція креслить лінію від поточної позиції до вказаної точки, але не включає її
MoveToEx	Функція оновлює поточну позицію вказаної точки та необов'язково повертає попередню позицію
PolyBezier	Функція рисує одну або декілька кривих Безьє
PolyBezierTo	Функція рисує одну або декілька кривих Безьє
PolyDraw	Функція рисує ряд відрізків прямих і криві Безьє
Polyline	Функція рисує ряд відрізків прямих, підключаючи точки у вказаному масиві
PolylineTo	Функція рисує одну або декілька прямих ліній
PolyPolyline	Функція рисує багатократний ряд зв'язаних відрізків
SetArcDirection	Функція встановлює напрям рисування, яке використовується для функцій прямокутника і дуги

### 2.3.1. Лінії та криві

Для рисування однієї *лінії* використовується функція `LineTo`, в якій безпосередні параметри можна вказувати як у процедурі, так і окремо – в сегменті даних, наприклад:

```
.data
nXEnd dd 1000
nYEnd dd 150
...
.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
; оброблення повідомлення WM_PAINT
invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури та
; заповнення структури ps
mov hdc, eax ; збереження контексту
invoke GetClientRect,hwnd, ADDR rect ; занесення в структуру rect
; характеристик вікна
invoke LineTo, \ ; функція рисування лінії
hdc, \ ; дескриптор контексту пристрою
nXEnd, \ ; x-координата закінчення лінії
nYEnd ; y-координата закінчення лінії
```

Для рисування *ломаної лінії* необхідно задати парі координат  $x$  та  $y$  точки, через які проводяться лінії з'єднання, та вказати функцію `Polyline`, наприклад:

```
.data
poln dna <40,50>, <40,200>, <150,200>, <150,50>, <260,50> ; координати
...
```

invoke Polyline, hdc, addr poln, \ адреса масиву, що містить кінцеві точки  
5 ; число пунктів у масиві

Для рисування *дуги* використовується функція з параметрами, наприклад:

```
invoke Arc, hdc, \ ; дескриптор пристрою  
50, \ ; X-координата квадрата лівого верхнього кута, в який буде вписана дуга  
50, \ ; Y-координата квадрата лівого верхнього кута, в який буде вписана дуга  
250, \ ; X-координата квадрата правого нижнього кута, в який буде вписана дуга  
250, \ ; Y-координата квадрата правого нижнього кута, в який буде вписана дуга  
50, 50 \ ; X-, Y-координати точки початку дуги  
170, 180 \ ; X-, Y- координати точки кінця дуги
```

Функція *кривих Безьє* відображує одну або декілька кривих, потребує додаткового масиву координат. Причому для однієї кривої необхідні три координати: точки початку й кінця та ще хоча б одна контрольна, наприклад:

```
.data  
pt dd 100,100,200,800,1000,400,100,600 ; пари координат  
...  
invoke PolyBezier, \ ; функція рисування кривих Безьє  
hdc, \ ; дескриптор контексту пристрою  
addr pt, \ ; адреса масиву пар координат  
4 ; кількість пар координат у масиві
```

Наприклад, у наведених рядках програми масив `pt` містить: координати точки 100, 100 – координати  $x$  та  $y$  точки початку кривої; координати 200, 800 – координати  $x$  та  $y$  контрольної точки кривої; координати 1000, 400 – координати  $x$  та  $y$  точки закінчення *однієї* кривої; координати 100, 600 – координати  $x$  та  $y$  точки закінчення *другої* кривої.

Для рисування *дуги* використовуються функції `Arc`, `ArcTo` та `AngleArc`.

Функція **Arc** має такі параметри:

```
invoke Arc, \ ; функція рисування дуги  
hdc, \ ; дескриптор контексту пристрою  
x1, y1, \ ; верхній лівий кут обмежувального прямокутника  
x2, y2, \ ; правий нижній кут обмежувального прямокутника  
x3, y3, \ ; початкова точка дуги  
x4, y4 ; кінцева точка дуги
```

У Windows NT напрям дуги визначається відповідним атрибутом у контексті пристрою, значення якого можна набути викликом функції `GetArcDirection` або встановити викликом функції `SetArcDirection`. Значення `AD_COUNTERCLOCKWISE` встановлює режим рисування



проти годинникової стрілки, а значення `AD_CLOCKWISE` – за годинниковою стрілкою. За умовчанням в контексті пристрою використовується значення `AD_COUNTERCLOCKWISE`.

Функція `ArcTo` відрізняється від функції `Arc` тим, що вона проводить лінію з поточної позиції пера в задану початкову точку дуги, і лише після цього рисує дугу. Після завершення рисування функція переміщає поточну позицію пера в кінцеву точку дуги.

Функція `AngleArc`, що істотно відрізняється від двох попередніх способом рисування дуги, має такі параметри:

```
invoke AngleArc, ; функція рисування дуги
hdc,\ ; дескриптор контексту пристрою
int_x1, int_y1,\ ; координати центра круга
DWORD Radius,\ ; радіус круга
FLOAT eStartAngle,\ ; початковий кут дуги в градусах
FLOAT eSweepAngle ; довжина дуги в градусах
```

### 2.3.2. Замкнуті фігури

Указана послідовність дій у мінімальному варіанті для рисування *квадрата*, наприклад, може бути такою:

```
.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
; оброблення повідомлення WM_PAINT
invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури та
; заповнення структури ps
mov hdc, eax ; збереження контексту
invoke GetClientRect,hwnd, ADDR rect ; занесення в структуру rect
; характеристик вікна
invoke Rectangle \ ; функція рисування квадрата
hdc \ ; дескриптор контексту пристрою
40, 45 \ ; x-, y-координати верхнього лівого кута прямокутника
250, 255 \ ; x-, y-координати нижнього правого кута прямокутника
invoke EndPaint,hwnd, ADDR ps ; закінчення рисування - повернення кон-
тексту
.ELSE
```

Наведений фрагмент виводить квадрат з використанням інструментів за умовчанням – чорне перо завтовшки 1 піксел і білу щітку для зафарбування внутрішньої ділянки фігури.

Прямокутник можна нарисувати з використанням структури даних `RECT`:

```
typedef struct tagRECT {
```

LONG left , LONG top ; координати лівого верхнього кута прямокутника  
 LONG right, LONG bottom ; координати правого нижнього кута  
 } RECT;

У табл. 2.2 наведено функції Win32 API, які використовують структуру RECT.

Таблиця 2.2 – Функції для роботи із структурами типу RECT

Функція	Опис
SetRect(&rect, xLeft, yTop, xRight, yBottom)	Установити координати прямокутника в задані значення
OffsetRect(&rect, dX, dY)	Перемістити прямокутник по осі X на величину dX та по осі Y на величину dY
InflateRect(&rect, dX, dY)	Зменшити або збільшити ширину і висоту прямокутника
SetRectEmpty(&rect)	Установити всі поля структури rect в нуль
CopyRect(&DestRect, &SrcRect)	Скопіювати один прямокутник в інший
IntersectRect(&DestRect, &SrcRect1, &SrcRect2)	Отримати перетин двох прямокутників
UnionRect(&DestRect, &SrcRect1, &SrcRect2)	Отримати об'єднання двох прямокутників
bEmpty = IsRectEmpty(&rect)	Визначити, чи є прямокутник порожнім
blnRect = PtInRect(&rect, point)	Визначити, чи міститься точка point усередині прямокутника rect

Функція Ellipse рисує *еліпс*. Вона може бути записана таким чином:

```
invoke Ellipse, \ ; функція рисування еліпса
hdc, \ ; дескриптор контексту пристрою
450, 350\ ; x-, y-координати верхнього лівого кута прямокутника
250, 95 ; x-, y-координати нижнього правого кута прямокутника
```

**Закруглений прямокутник** будується за допомогою функції RoundRect. Він може бути записаний таким чином:

```
invoke RoundRect, \ ; функція рисування закругленого прямокутника
hdc, \ ; дескриптор контексту пристрою
50, 50, \ ; x-, y-координати верхнього лівого кута прямокутника
500, 500, \ ; x-, y-координати нижнього правого кута прямокутника
100, 100 ; ширина та висота еліпса для рисування кута
```

На рис. 2.1 наведено відображення прямокутника з закругленими кутами з товщиною ліній в 1 піксел чорним кольором.

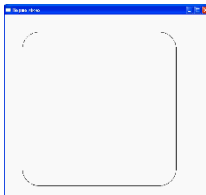


Рис. 2.1. Відображення прямокутника з закругленими кутами

Для роботи з прямокутниками використовуються такі функції:

- **CopyRect** – копіює координати одного прямокутника в інший;
- **EqualRect** – визначає ідентичність двох заданих прямокутників, порівнюючи координати їх верхніх лівих і нижніх правих кутів;
- **InflateRect** – збільшує або зменшує ширину та висоту вказаного прямокутника. Функція **InflateRect** додає параметр  $dx$  до лівого і правого країв прямокутника, а параметр  $dy$  – до нижнього і верхнього країв. Значення параметрів  $dx$  і  $dy$  є знаковими величинами: позитивні значення збільшують ширину і висоту прямокутника, а негативні значення їх зменшують;
- **IntersectRect** – обчислює перетин двох початкових прямокутників і поміщає координати прямокутника перетину в третій прямокутник. Якщо початкові прямокутники не перетинаються, в прямокутник призначення поміщається порожній (у якого всі координати дорівнюють нулю) прямокутник;
- **IsRectEmpty** – визначає, чи є заданий прямокутник порожнім. Порожнім є прямокутник, що не має площі. Тобто координата правої сторони менша від координат лівої сторони або дорівнює їм, або координата нижньої сторони менша від координат верхньої сторони або дорівнює їм;
- **OffsetRect** – переміщає прямокутник на вказаний зсув;
- **PtInRect** – визначає, чи знаходиться задана точка усередині вказаного прямокутника. Точка, яка лежить на лівій або верхній стороні прямокутника, також вважається точкою, що лежить усередині прямокутника. Точка, яка лежить на правій або на нижній стороні прямокутника, вважається точкою, що лежить поза ним;
- **SetRect** – встановлює координати вказаного прямокутника. Це еквівалентно присвоюванню нових значень відповідним членам структури типу **RECT**;

- **SetRectEmpty** – створює порожній прямокутник, в якому всі координати встановлені в нуль;
- **SubtractRect** – отримує координати прямокутника, визначуваного відніманням одного прямокутника з іншого;
- **UnionRect** – створює об'єднання двох прямокутників. Об'єднання – це найменший прямокутник, який містить обидва початкових прямокутники.

**Багатокутник** рисується за допомогою функції **Polygon**, приклад використання якої може бути таким:

```
.data
poln POINT <100,300>, <140,260>, <700,260>, <740,300>, <700,340>,\
<140,340> ; координати
...
invoke Polygon,\      ; функція рисування багатокутника
hdc,\                  ; дескриптор контексту пристрою
addr poln, \          ; адреса масиву координат точок
6                      ; число вершин у масиві
```

**Сегмент** еліпса утворюється з дуги, якщо її кінці з'єднати відрізком, званим хордою. Для рисування сегментів використовується функція **Chord**:

**BOOL Chord**(HDC hdc, int xLeft, int yTop, int xRight, int yBottom, int xStart, int yStart, int xEnd, int yEnd)

Функція приймає такий же набір параметрів, що й функція **Arc**. Початковий і кінцевий кути дуги задаються двома точками з координатами (xStart, yStart) та (xEnd, yEnd). Напрямок рисування дуги визначається так само, як і для функції **Arc**.

**Сектор** еліпса – це фігура, обмежена дугою та двома радіусами, проведеними до кінців дуги. Сектори рисують функцією **Pie**, прототип якої такий:

**BOOL Pie**(HDC hdc, int xLeft, int yTop, int xRight, int yBottom, int xStart, int yStart, int xEnd, int yEnd)

Функція приймає такий же набір параметрів, що й функція **Arc**. Напрямок рисування дуги та вигляд отримуваної фігури визначаються так само, як і для функції **Arc**.

## 2.4. Перо

Перо застосовується для рисування ліній. За умовчанням використовується суцільна чорна лінія завтовшки 1 піксель.

Для задоволення інших властивостей Win32 GDI дозволяє створювати об'єкти логічного пера. *Логічним пером* є опис вимог до пера з боку застосування. Драйвер графічного пристрою може підтримувати власні структури даних, що визначають реалізацію логічного пера – такі внутрішні об'єкти називаються *фізичним пером*.

### 2.4.1. Стандартне перо

У Win32 GDI за умовчанням використовується суцільне чорне перо завтовшки 1 піксел з індексом `BLACK_PEN`. Існує можливість використання суцільного білого пера товщиною 1 піксел (індекс `WHITE_PEN`), а порожнє перо має ім'я `NULL_PEN`.

Для отримання дескриптора стандартного об'єкта потрібно викликати функцію `GetStockObject`, передавши їй індекс стандартного об'єкта.

Набувши значення дескриптора, потрібно вибрати об'єкт пера в контекст пристрою:

```
SelectObject, \ ; виберемо в контекст  
hdc, hPen ; нове перо
```

Після цього всі функції, які рисують лінії, використовуватимуть `WHITE_PEN`.

Бажане збереження попереднього дескриптора об'єкта і повернення його в контекст пристрою після завершення операцій рисунка з новим об'єктом.

Для зміни кольору пера передбачена функція `SetDCPenColor`, що має такий прототип:

```
SetDCPenColor, \ ; зміна кольору пера  
hdc, ; \  
crColor ;
```

Параметр `crColor` задає новий колір пера DC. Функція повертає попередній колір пера DC.

### 2.4.2. Просте перо

Для рисунка переривистих або товстих ліній потрібно використовувати нестандартні об'єкти логічного пера. Просте перо створюється викликом функції `CreatePen` або `CreatePenIndirect`.

```
Функція CreatePen має такий прототип:  
CreatePen, \ ; створення логічного пера  
fnPenStyle, \ ; стиль пера
```

int nwidth, \ ; ширина пера в логічних одиницях  
crColor ; колір пера

Функція **CreatePen** створює логічне перо і має 3 параметри.

Перший параметр указує стиль пера:

PS\_SOLID – суцільна лінія;

PS\_DASH – пунктирна лінія;

PS\_DOT – лінія з точок;

PS\_DASHDOT – перо має передуючі лінії та точки;

PS\_DASHDOTDOT – перо має передуючі лінії та подвійні точки;

PS\_NULL – перо невидиме;

PS\_INSIDEFRAME – суцільна лінія. Застосовується до геометричних фігур.

Перший параметр задає стиль пера, що визначає порядок проходження пікселів і розташування лінії.

Другий параметр функції **CreatePen** задає товщину лінії в *логічних одиницях*. Параметру *nwidth* можна передати нульове значення. В цьому випадку використовуватиметься перо завтовшки 1 піксел незалежно від режиму відображення.

Стильові (переривисті) лінії можна рисувати тільки з товщиною 1 піксел. Точки в стильових лініях зображуються трьома пікселями.

Третій параметр функції **CreatePen** має спеціальний тип **COLORREF** і визначає колір пера. Функція повертає дескриптор створеного пера, який потім слід завантажити в контекст пристрою. Цей тип описує 32-бітове дане. Зручніше використовувати структуру **RGBQUAD**, що належить до 16-кольорового режиму. У цьому режимі значення компонентів можуть становити від 0 до 255 (повна яскравість компонента).

Приклади використання структури **RGBQUAD**:

```
brown RGBQUAD <128,128,0> ; червоно-зелений (коричневий)  
yellow RGBQUAD <255,255,0> ; яскравий червоно-зелений (жовтий)  
cyan RGBQUAD <0,128,128> ; синьо-зелений (бірюзовий)  
color1 RGBQUAD <192,192,192> ; ясно-сірий  
color2 RGBQUAD <192,220,192> ; ясно-салатовий  
color3 RGBQUAD <128,255,255> ; ясно-бірюзовий
```

Другий спосіб створення простого пера пов'язаний з викликом функції **CreatePenIndirect**:

**CreatePenIndirect**, \ ; створення логічного пера, яке визначається **LogPen**  
**LogPen** ;

Цій функції як параметр передається адреса структури типу **LogPen**:

```

struct LogPen
    style          ; стиль пера
    POINT width    ; товщина в логічних одиницях
    COLORREF color ; колір

```

Член структури `width` має тип `POINT`, але Windows використовує тільки величину `width.x` як товщину пера та ігнорує значення `width.y`.

Функції `CreatePen` та `CreatePenIndirect` не вимагають дескриптора контексту пристрою, поки не буде викликана функція `SelectObject`.

Закінчивши рисувати, потрібно видалити вибране перо. Якщо дескриптор не збережений, то можна викликати функцію `SelectObject` з її властивістю повертати дескриптор об'єкта, вибраного в контекст пристрою раніше.

Наприклад, застосуванню потрібні три нестандартні пера: червоне завтовшки 4 піксели, зелене завтовшки 6 пікселів і синє пунктирне. Спочатку потрібно визначити змінні для зберігання дескрипторів цих пер.

Самі пера можуть бути створені в процесі обробки повідомлення `WM_CREATE`. Фрагмент програми може бути таким:

```

.data
    hPen1 dd 0 ; комірка для збереження пера hPen1
    hPen2 dd 0 ; комірка для збереження пера hPen2
    hPen3 dd 0 ; комірка для збереження пера hPen3
    color1 RGBQUAD <255,0,0> ; структура color1 для завдання кольору
    color2 RGBQUAD <0,255,0> ; структура color2 для завдання кольору
    color3 RGBQUAD <0,0,255> ; структура color3 для завдання кольору
    ...
invoke CreatePen,\ ; створення пера
    PS_SOLID,4, dword ptr color1 ; завтовшки 4 піксели, червоний колір
    mov hPen1,eax ; збереження отриманого дескриптора пера
invoke CreatePen,\ ; створення пера
    PS_SOLID,6, dword ptr color2 ; завтовшки 6 пікселів, зелений колір
    mov hPen2,eax ; збереження отриманого дескриптора пера
invoke CreatePen,\ ; створення пера
    PS_DASH,1, dword ptr color3 ; пунктирне, 1 піксел, синій колір
    mov hPen3,eax ; збереження отриманого дескриптора пера

```

У процесі оброблення повідомлення `WM_PAINT` можна вибрати одне з цих пер в контекст пристрою і рисувати з його допомогою:

```

Selectobject, hdc, hpen1 ;
; ... функції рисування ліній
Selectobject, hdc, hpen2);
; ... функції рисування ліній
Selectobject, hdc, hpen3);

```

; ... функції рисування ліній

У процесі оброблення повідомлення WM\_DESTROY рекомендується видалити ці пера:

```
DeleteObject, hPen1 ;  
DeleteObject, hPen2 ;  
DeleteObject, hPen3 ;
```

Окрім такого сценарію застосування пера, можна створити перо в блоці оброблення повідомлення WM\_PAINT і видалити його перед викликом функції ENDPAINT або навіть після її виклику.

### 2.4.3. Розширене перо

Стильові лінії можна рисувати тільки завтовшки 1 піксел, а суцільні лінії рисують будь-якою товщиною, але вони завжди мають закруглені закінчення.

Для подолання цих обмежень у Win32 API з'явилася функція ExtCreatePen, що створює перо з розширеним набором атрибутів ([https://msdn.microsoft.com/en-us/library/windows/desktop/dd162705\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dd162705(v=vs.85).aspx)):

```
ExtCreatePen, \ ;  
DWORD dwPenStyle, \ ; тип, стиль та атрибути пера  
DWORD dwWidth, \ ; товщина  
CONST LOGBRUSH* lplb, \ ; атрибути щітки  
DWORD dwStyleCount, \ ; довжина масиву lpstyle  
CONST DWORD* lpStyle ; масив, якій задає правила чергування пікселів
```

Параметр dwPenStyle може набувати значення у вигляді комбінації прапорців, що визначають *тип пера*, його *стиль*, *тип завершення ліній* і *тип з'єднання ліній*. Можливі значення цих прапорців наведено в табл. 2.3 – 2.6.

Таблиця 2.3 – Типи розширеного пера

Прапорець типу	Опис
PS_COSMETIC	Косметичне перо. Товщина дорівнює 1 пікселю
PS_GEOMETRIC	Геометричне перо. Товщина – в логічних одиницях



Таблиця 2.4 – Стилі розширеного пера

Прапорець стилю	Опис
PS_ALTERNATE	Чергування «піксел – проміжок» для косметичного пера
PS_SOLID	Безперервна лінія
PS_DASH	Штрихова лінія
PS_DOT	Пунктирна лінія
PS_DASHDOT	Пунктирно-точкова лінія
PS_DASHDOTDOT	Відрізок і дві точки
PS_NULL	Лінія не рисується
PS_INSIDEFRAME	Безперервна лінія. Тільки для геометричного пера
PS_SERSTYLE	Чергування відрізків і проміжків, задається параметрами dwStyleCount и lpStyle

Усі стилі, окрім PS\_INSIDEFRAME, мають вирівнювання по центру.

Таблиця 2.5 – Тип завершення, використовуваний тільки для геометричного пера

Прапорець типу	Опис
PS_ENDCAP_ROUND	Закруглене закінчення, коли до лінії додається половина круга
PS_ENDCAP_SQUARE	Квадратне закінчення, коли до лінії додається половина квадрата
PS_ENDCAP_FLAT	Плоске закінчення

Таблиця 2.6 – Тип з'єднання, використовуваний тільки для геометричного пера

Прапорець типу	Опис
PS_JOIN_BEVEL	Усічене з'єднання
PS_JOIN_MITER	Загострене з'єднання
PS_JOIN_ROUND	Закруглене з'єднання

#### 2.4.4. Косметичне перо

Косметичне перо можна використовувати тільки з товщиною 1 піксел. При спробі задати велику товщину Windows використовує стиль PS\_SOLID. Для стилів PS\_DASH, PS\_DOT, PS\_DASHDOT та

`PS_DASHDOTDOT` косметичні пера рисують такі самі лінії, як і прості пера. Проте, на відміну від простого пера, вони завжди рисують у прозорому режимі змішування фону, навіть якщо в контексті пристрою встановлений режим `OPAQUE`.

Для стилю `PS_ALTERNATE` косметичне перо рисує «справжню» точкову лінію, яку утворюють точки розміром 1 піксел і проміжки між точками також розміром 1 піксел.

Для використання стилю `PS_USERSTYLE` необхідні два додаткових параметри: `dwStyleCount`, що задають кількість елементів у масиві з адресою `lpStyle`. Перший елемент масиву містить довжину першого відрізка, другий – довжину проміжку, третій – довжину другого відрізка і так далі. При цьому одна одиниця довжини відповідає трьом пікселам замість одного.

#### *2.4.5. Геометричне перо*

Геометричне перо рисує лінії, які можуть розрізнятися товщиною, стилем, заливкою, типом завершення і типом з'єднання.

Товщина геометричного пера задається в логічних координатах. Якщо в простому пері одиниця довжини, відповідно зображення точки, дорівнює 3 пікселям, то в геометричному пері одиниця довжини в режимі відображення `MM_TEXT` дорівнює 1 пікселю. Але із збільшенням товщини лінії пропорційно збільшується і довжина точки, так само, як і довжина інших відрізків, що сановають лінію. Також ці розміри можуть змінюватися відповідно до світових перетворень або режиму відображення.

На відміну від простого пера потовщене геометричне перо рисує лінії відповідно до свого стилю.

Геометричне перо за умовчанням має закінчення, визначуване стилем `PS_ENDCAP_ROUND`. Але тип закінчення можна змінити, якщо при виклику функції `ExtCreatePen` передати перший параметр з додаванням прапорця з табл. 2.7.

Будь-яку замкнуту фігуру, наприклад прямокутник або еліпс, рисують у Windows з використанням двох графічних об'єктів, вибраних у контекст пристрою. Першим об'єктом є поточне перо, яким обводиться контур фігури, а другим об'єктом – поточна щітка, використовувана для заливки (зафарбування) внутрішньої ділянки фігури.

Таблиця 2.7 – Можливі значення поля `lbStyle`

Значення	Опис
<code>BS_SOLID</code>	Суцільна щітка
<code>BS_HATCHED</code>	Штрихова щітка
<code>BS_HOLLOW</code>	Пуста щітка (заливки немає)
<code>BS_NULL</code>	Те саме, що й <code>BS_HOLLOW</code>
<code>BS_PATTERN</code>	Растрова щітка, рисунок якої визначений растровим зображенням у пам'яті
<code>BS_PATTERN8X8</code>	Те саме, що й <code>BS_PATTERN</code>
<code>BS_DIBPATTERN</code>	Растрова щітка, рисунок якої задається апаратно-незалежним растровим зображенням ( <code>DIB – device-independent bitmap</code> ), дескриптор якого міститься в полі <code>lbHatch</code>

## 2.5. Щітки

Щітка – це растр розміром  $8 \times 8$  пікселів. При зафарбуванні ділянки вона дублюється в горизонтальному та вертикальному напрямі.

Логічна щітка описує вимоги, що ставляться до заливки з боку застосування. Ці вимоги не завжди збігаються з можливостями фізичних пристроїв. Драйвери пристроїв підтримують власні структури даних, що визначають реалізацію логічної щітки. Такі внутрішні об'єкти називаються фізичними щітками.

Для дескрипторів логічних щіток зарезервований тип `HBRUSH` (*handle to a brush*).

Значення дескриптора отримують викликом відповідної функції. Створені щітки вибираються в контекст пристрою за допомогою функції `SelectObject`, після чого заливка всіх замкнутих фігур здійснюється вибраною щіткою. Непотрібну щітку необхідно видалити за допомогою функції `DeleteObject`.

### 2.5.1. Стандартні щітки

Стандартні щітки GDI наведено в табл. 2.8.

Так само, як і перо DC, щітка DC дозволяє змінювати свій колір після її вибору в контекст пристрою. Для цього призначено функцію `SetDCBrushColor`.

Щоб отримати дескриптор стандартної щітки, досить викликати функцію `GetStockObject` з однією з констант.

Таблиця 2.8 – Стандартні щітки

Індекс щітки	Опис
BLACK_BRUSH	Чорна щітка
DKGRAY_BRUSH	Темно-сіра щітка
DC_BRUSH	Щітка DC — суцільна кольорова щітка; за умовчанням має білий колір; колір може бути змінений функцією <code>SetDCBrushColor</code>
GRAY_BRUSH	Сіра щітка
HOLLOW_BRUSH	Порожня щітка (заливки немає)
LTGRAY_BRUSH	Ясно-сіра щітка
NULL_BRUSH	Те саме, що і HOLLOW_BRUSH
WHITE_BRUSH	Біла щітка, яка використовується за умовчанням

### 2.5.2. Щітка користувача

Windows містить функції, що створюють призначені для користувача щітки таких типів: *суцільні щітки*, *штрихові щітки* та *растрові щітки*.

### 2.5.3. Суцільні щітки

Суцільна щітка створюється викликом функції `CreateSolidBrush`:  
`CreateSolidBrush, \;`

`crColor`; значення кольору щітки

Її єдиному параметру `crColor` передається колір щітки у вигляді значення типу `COLORREF`. Це значення задається за допомогою макросу `RGB` або `PALETTEINDEX`. Перший варіант використовується, якщо пристрій виведення підтримує повний діапазон кольорів, визначуваний 24-бітовим `RGB`-значенням. Другий варіант необхідно використовувати, якщо застосування працює з логічною палітрою. В останньому випадку Windows перетворить запитаний в макросі `PALETTEINDEX` колір в найбільш відповідний індекс палітри.

### 2.5.4. Штрихові щітки

Штрихова щітка створюється викликом функції `CreateHatchBrush`:

`CreateHatchBrush, \`; виклик штрихової щітки  
`int fnStyle \`; стиль штрихування

COLORREF clrref ; колір

Параметр `fnStyle`, який задає стиль штрихування, може набувати значення `HS_HORIZONTAL`, `HS_VERTICAL`, `HS_BDIAGONAL`, `HS_FDIAGONAL`, `HS_CROSS` та `HS_DIAGCROSS`.

Колір основних пікселів задається параметром `clrref`, а колір фонових пікселів визначається атрибутом кольору фону графічних елементів у контексті пристрою, який можна встановлювати за допомогою функції `SetBKColor`. Фонові пікселі виводяться тільки в тому випадку, якщо встановлений режим змішування фону `OPAQUE`.

GDI дозволяє також встановлювати базову точку штрихової щітки за допомогою функції `SetBrushOrgEX`:

`SetBrushOrgEX, \` ; встановлення базової точки штрихової щітки

`hdc \` ; контекст пристрою

`int nXOrg \` ; x-координата нової точки

`int nYOrg \` ; y-координата нової точки

`LPPPOINT lppt` ; адреса структури з колишніми координатами точки

Базова точка (`nXOrg`, `nYOrg`), що задається в системі координат пристрою, визначає прив'язку лівого верхнього пікселя штрихового візерунка. Параметр `lppt` містить адресу структури типу `POINT`, в якій запам'ятовується колишня базова точка. За умовчанням координати базової точки щітки дорівнюють (0, 0).

### 2.5.5. Растрові щітки

Растрова щітка створюється за допомогою функції `CreatePatternBrush`:

`CreatePatternBrush, \` ;

`HBITMAP hbmp` ;

Параметр `hbmp` містить дескриптор растрового об'єкта (*bitmap*) GDI.

Уставити растровий об'єкт можна за допомогою такого фрагмента:

`LoadImage(0, "Star.bmp", IMAG_EBITMAP, 0, 0, LR_LOADFROMFILE;`

`CreatePatternBrush, \` ;

`hBmp` ;

`SelectObject, hDC, hBrush` ;

`Rectangle, hDC, 32, 32, 288, 160);`

`DeleteObject, hDC` ;

Функція `LoadImage` у цьому фрагменті завантажує апаратно-незалежний растр з файлу `Star.bmp`, потім перетворює його до формату ра-

стового об'єкта GDI і повертає дескриптор цього об'єкта `hBmp`. Отриманий дескриптор передається функції `CreatePatternBrush`, яка створює растрову щітку. Щітка `hBrush` вибирається в контекст пристрою і після цього використовується для заливки внутрішньої ділянки прямокутника, що рисується функцією `Rectangle`.

### 2.5.6. Структура **LOGBRUSH** і функція **CreateBrushIndirect**

Win32 GDI надає ще одну функцію — `CreateBrushIndirect`, що дозволяє створити логічну щітку будь-якого з трьох розглянутих вище типів:

```
CreateBrushIndirect(\); створення лог. щітки, яка визначається LOGBRUSH
CONST, \          ;
LOGBRUSH* lplb  ; адреса структури для параметрів щітки
```

Параметр `lplb` – це адреса структури типу `LOGBRUSH`, у якій задаються параметри щітки:

```
LOGBRUSH struc, \ ; структура для створення розширеного пера
    UINT  lbStyle, \ ; стиль щітки
    COLORREF lbColor, \ ; колір щітки
    LONG  lbHatch  ; стиль штрихування
LOGBRUSH ends
```

Поле `lbStyle` задає стиль щітки і може мати одне із значень, наведених у табл. 2.7.

Інтерпретацію поля `lbcolor` наведено в табл. 2.9.

Таблиця 2.9 – Інтерпретація поля `lbcolor`

Значення поля <code>lbStyle</code>	Інтерпретація поля <code>lbColor</code>
<code>BS_HOLLOW</code> або <code>BS_PATTERN</code>	Ігнорується
<code>BS_HATCHED</code> або <code>BS_SOLID</code>	Значення типу <code>COLORREF</code>
<code>DIB_PAL_COLORS</code> або <code>DIB_RGB_COLORS</code>	Молодша частина слова <code>lbcolor</code> повинна мати значення <code>DIB_PAL_COLORS</code> або <code>DIB_RGB_COLORS</code> . Перше значення вказує на те, що в масиві <code>bmicolors</code> , що є членом структури <code>BITMAPINFO</code> , містяться 16-розрядні індекси в логічній палітрі, друге значення – на те, що в масиві <code>bmiColors</code> містяться явні RGB-коди кольорів

Поле `lbHatch` задає стиль штрихування. Його інтерпретація також залежить від значення поля `lbStyle` і пояснюється в табл. 2.10.

Таблиця 2.10 – Інтерпретація поля `lbHatch`

Значення поля <code>lbStyle</code>	Інтерпретація поля <code>lbHatch</code>
<code>BS_SOLID</code> або <code>BS_HOLLOW</code>	Ігнорується
<code>BS_PATTERN</code>	Дескриптор растрового об'єкта GDI (bitmap'a)
<code>BS_HATCHED</code>	Одне із значень <code>HS_HORIZONTAL</code> , <code>HS_VERTICAL</code>
<code>BS_DIBPATTERN</code>	Дескриптор упакованого DIB-растра
<code>BS_DIBPATTERNPT</code>	Указівник на упакований DIB-растр

Структура `LOGBRUSH` також використовується при створенні розширеного пера.

## 2.6. Зміна характеристик графічних інструментів

Характеристики пера, що знаходиться в контексті пристрою, визначають вигляд ліній, якими рисуються, наприклад, прямокутники або еліпси (а також і просто лінії). У контексті пристрою зберігається дескриптор пера за умовчанням, яке має **чорний колір** і **товщину 1 піксел**. *Змінити характеристики існуючого пера не можна, проте можна створити нове перо (або будь-яку кількість нових пер) з необхідними характеристиками – кольором і товщиною.* Створивши нове перо, слід помістити в контекст його дескриптор, після чого саме воно використовуватиметься програмами GDI при виведенні на екран ліній або геометричних фігур.

При роботі з новими інструментами слід дотримуватися певної стандартної послідовності дій:

- створення нового інструменту із заданими характеристиками за допомогою, наприклад, функції `Windows CreatePen` (для пера) або `CreateSolidBrush` (для щітки) і отримання його дескриптора;

- завантаження (вибір) в контекст пристрою дескриптора створеного інструменту з обов'язковим збереженням дескриптора аналогічного інструменту за умовчанням; ця дія виконується за допомогою функції `SelectObject`;

- рисування новим інструментом;

- знищення створеного інструменту функцією DeleteObject.

Створення нового інструменту можна виконати в будь-якому місці програми; найприродніше це зробити при обробці повідомлення WM\_CREATE для головного вікна або того вікна, для якого готуються нові інструменти.

Функції вибору і рисування вимагають як перший параметр указівку дескриптора контексту пристрою, тому ці функції зазвичай викликаються в процесі оброблення повідомлення WM\_PAINT після отримання контексту пристрою функцією BeginPaint. *У контексті пристрою зберігається тільки один дескриптор для кожного інструменту. Повторний вибір у контекст нового дескриптора не вимагає збереження попереднього.* Важливо зберегти тільки дескриптор початкового інструменту (інструменту за умовчанням). Саме його слід відновити в контексті пристрою перед закриттям контексту функцією EndPaint.

Видалення створених інструментів можливе в будь-якому місці, проте природніше це виконати у функції оброблення повідомлення WM\_DESTROY.

Фрагмент програми використання нового пера може бути таким:

```
. data
hPen1  dd 0      ; дескриптор створюваного пера
hOldPen dd 0      ; дескриптор початкового пера
color3  RGBQUAD <128,255,255> ; ясно-бірюзовий
. code
...
WndProc proc hWnd:HWND, uMsg:UINT, \
    wParam:WPARAM, lParam:LPARAM
LOCAL hdc:HDC      ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
PAINTSTRUCT
    LOCAL rect:RECT ; резервування стека під структуру RECT
    .IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
        invoke DeleteObject, hPen1 ; видаляє Handle з пам'яті і звільняє ресу-
        рси
        invoke DeleteObject, hOldPen ; видаляє Handle з пам'яті і звільняє ресу-
        рси
        invoke PostQuitMessage, NULL ; передача повідомлення WM_Quit
    .ELSEIF uMsg==WM_CREATE ; оброблення повідомлення
        invoke CreatePen, \
            PS_SOLID,40,\ ; створення пера
            dword ptr color3 ; завтовшки 40 пікселів
            mov hPen1, eax ; збереження отриманого дескриптора пера
```



```

.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
    invoke BeginPaint \ ; підготування вікна до розфарбовування
        hdc, \ ; контекст пристрою
        addr ps ; заповнення інформацією для розфарбовування
    mov     hdc, eax ; збереження дескриптора контексту в hdc
invoke SelectObject, \ ; виберемо в контекст
    hdc, \ ; повернення дескриптора
    hPen1 ; нове перо
    mov     hOldPen, eax ; збереження початкового пера
    invoke RoundRect, hdc, \ ; функція рисування закругленого прямокутника
        50,50, \ ; x- та y-координата верхнього лівого кута прямокутника
        500,500, \ ; x- та y-координата нижнього правого кута прямокутника
        100,100 ; ширина та висота еліпса для рисування кута
    invoke EndPaint, \ ; віддача контексту
        hWnd, addr ps ; системі
.ELSE
    invoke DefWindowProc, hWnd, uMsg, wParam, lParam ; обробка
    ret
.ENDIF

```

На рис. 2.2 наведено відображення прямокутника з закругленими кутами з товщиною в 40 пікселів ясно-бірюзового кольору.

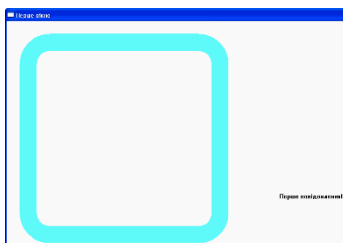


Рис. 2.2. Відображення прямокутника з закругленими кутами з товщиною в 40 пікселів

У процесі створення вікна (викликом функції Windows CreateWindow) Windows посилає у застосунок повідомлення WM\_CREATE. На цьому етапі роботи застосунку створюються всі використовувані надалі пера і шрифти; у наведеному вище фрагменті у відповідь на повідомлення WM\_CREATE створюється (викликом функції Windows CreatePen) товсте суцільне перо світло-сірого кольору. Функція CreatePen повертає в регістрі EAX дескриптор створеного інструменту, який зберігається в заздалегідь передбаченому для цього елементі пам'яті hPen1.

У відповідь на повідомлення WM\_PAINT викликом функції BeginPaint отримуємо у Windows контекст пристрою і зберігаємо його дескриптор. Далі за допомогою універсальної функції SelectObject, призначеної для роботи з будь-якими інструментами (перами, щітками, шрифтами), завантажуюмо в контекст пристрою дескриптор створеного пера. Функція SelectObject, вибравши в контекст новий інструмент, повертає в регістрі EAX дескриптор того інструменту, який зберігався до цього в контексті. Нам він знадобиться надалі, тому ми зберігаємо його в осередку hOldPen.

Починаючи з цього моменту, будь-які геометричні фігури, що виводяться у вікно застосунку, рисуватимуться тим пером, дескриптор якого був вибраний в контекст пристрою.

Нарисувавши те, що потрібно (у наведеному фрагменті всього один прямокутник), за допомогою тієї ж функції SelectObject вибираємо в контекст збережений раніше дескриптор початкового пера (пера за умовчанням), викликом функції EndPaint повертаємо системі узятий у неї на якийсь час контекст пристрою і завершуємо на цьому обробленняповідомлення WM\_PAINT.

Як уже наголошувалося, перед своїм завершенням застосунок зобов'язаний звільнити всі зайняті ним ресурси Windows, зокрема, знищити створені інструменти. Найзручніше це виконати у відповідь на повідомлення WM\_DESTROY. Для знищення будь-яких інструментів можна використати універсальну функцію Windows DeleteObject.

Фрагмент програми використання нової щітки для зарисовування фону графічної фігури може бути таким:

```
.data ; директива визначення даних
ClassName db "Первый класс",0
AppName db "Перше вікно",0
Hello db "Перше повідомлення!!!",0
color3 RGBQUAD <128,255,255> ; ясно-бірюзовий
hPen1 dd 0
hOldPen dd 0
.data? ; директива невизначених даних
hInstance HINSTANCE ?
blueBrush dd ? ; комірка для параметрів щітки
...
WndProc proc hWnd:HWND, uMsg:UINT, \
    wParam:WPARAM, lParam:LPARAM
LOCAL hdc:HDC ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
PAINTSTRUCT
LOCAL rect:RECT ; резервування стека під структуру RECT
```

```

.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
    invoke DeleteObject, hPen1 ; видаляє Handle з пам'яті і звільняє ресурси
    invoke DeleteObject, hOldPen ; видаляє Handle з пам'яті і звільняє ресурси
    invoke PostQuitMessage, NULL ; передача повідомлення WM_Quit
.ELSEIF uMsg==WM_CREATE ; оброблення повідомлення
    WM_CREATE
    invoke CreatePen, \ ; створення пера
        PS_SOLID, 40, \ ; завтовшки 40 пікселів
        dword ptr color3 ; ясно-бірюзовий
        mov hPen1, eax ; збереження отриманого дескриптора пера
.ELSEIF uMsg==WM_PAINT ; якщо повідомлення про перерисовування
    invoke BeginPaint, hWnd, ADDR ps ; виклик підготовчої процедури
        mov hdc, eax
invoke SelectObject, \ ; виберемо в контекст
        hdc, hPen1 ; нове перо
mov hOldPen, eax ; збережемо початкове перо
    invoke GetClientRect, hWnd, ADDR rect ; занесення до структури rect ха-
    рактеристик вікна
    invoke DrawText, hdc, ; рисування тексту
        ADDR Hello, ; указівник на текстовий рядок
        -1, ; функція сама визначить довжину рядка
        ADDR rect, DT_SINGLELINE or DT_CENTER or DT_VCENTER

invoke CreateHatchBrush, \ ; створення щітки для зарисовування фігури
        HS_DIAGCROSS, \ ; стиль – штриховка під 45 градусів зліва та
        справа
        dword ptr color3 ; значення кольору
        mov blueBrush, eax ; збереження параметрів щітки
invoke SelectObject, \ ; вибір об'єкта з параметрами
        hdc, \ ; контекст пристрою
        blueBrush ; параметри щітки
    invoke Ellipse, \ ; функція рисування еліпса
        hdc, \ ; дескриптор контексту пристрою
        40, 220, \ ; x- та y-координата верхнього лівого кута прямокутника
        250, 95 ; x- та y-координата нижнього правого кута прямокутника
    invoke EndPaint, hWnd, ADDR ps ; закінчення рисування
.ELSE ; інакше (директива IF)
    invoke DefWindowProc, hWnd, uMsg, wParam, lParam ; оброблення
        ret ; повернення з процедури
.ENDIF ; закінчення директиви IF
    xor eax, eax ; заповнення нулями
    ret ; повернення управління ОС
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

Функція `CreateHatchBrush` створює щітку для зарисовування геометричної фігури. Другий параметр може мати такі значення:

`HS_BDIAGONAL` – штриховка під  $45^\circ$  знизу зліва направо  
`HS_CROSS` – штриховка горизонтально та вертикально  
`HS_DIAGCROSS` – штриховка під  $45^\circ$  знизу зліва та справа  
`HS_FDIAGONAL` – штриховка під  $45^\circ$  вгору зліва направо  
`HS_HORIZONTAL` – штриховка горизонтальна  
`HS_VERTICAL` – штриховка вертикальна

На рис. 2.3 наведено результат виконання фрагмента програми з виведенням еліпса з використанням кольорового пера шириною 40 пікселів та зарисовуванням еліпса відповідною штриховкою.

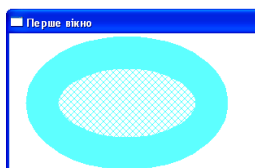


Рис. 2.3. Відображення еліпса з використанням кольорового пера та відповідної штриховки

Для організації виведення зображення у вікно програми з меню є два шляхи:

– використання структури `WNDCLASSEX` з фрагментом коду:

```
.data
    MenuName db "FirstMenu",0
    ...
.code
    mov     wc.lpszMenuName, OFFSET MenuName
```

– використання параметра хендла меню в функції `CreateWindowEx` з фрагментом коду:

```
.data
    MenuName db "FirstMenu",0
    hMenu HMENU ?
    ...
.code
    invoke LoadMenu, hInst, OFFSET MenuName
        mov     hMenu, eax
    invoke CreateWindowEx, \ ; створення вікна з розширеним стилем
        NULL, \ ; додатковий стиль
        ADDR ClassName, \ ; адреса імені класу
        ADDR AppName, \ ; адреса імені вікна
        WS_OVERLAPPEDWINDOW, \ ; базовий стиль
        CW_USEDEFAULT, \ ; горизонтальна координата вікна
```

CW\_USEDEFAULT, \ ; вертикальна координата вікна  
 CW\_USEDEFAULT, \ ; ширина вікна  
 CW\_USEDEFAULT, \ ; висота вікна  
 NULL, \ ; дескриптор батьківського вікна  
 hMenu, \ ; **дескриптор меню**  
 hInstance, \ ; дескриптор програми  
 NULL

У програмі 2.1 наведено програму відображення еліпса та виведення відповідного повідомлення з використанням файлу ресурсів з програмі 2.2.

### Програма 2.1:

```

.686 ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер

WinMain proto :DWORD,;DWORD,;DWORD,;DWORD

include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\include\user32.inc ; файли інтерфейсу ...
include \masm32\include\kernel32.inc ; файли систем. функцій застосунків...
include \masm32\include\gdi32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib c:\masm32\lib\gdi32.lib

.data ; директива визначення даних
  ClassName db "SimpleWinClass",0
  AppName db "Результат выполнения программы",0
  MenuName db "FirstMenu",0
  About_string db "Test menu",0
  color3 RGBQUAD <128,255,255> ; ясно-бірюзовий
  hPen1 dd 0
  hOldPen dd 0
  blueBrush dd ? ; комірка для параметрів щітки

.data?
  hInstance HINSTANCE ?
  CommandLine LPSTR ?

.const
  IDM_FUNC equ 1
  IDM_EXIT equ 2
  IDM_ABOUT equ 3
  IDI_ICON equ 22
  
```

```
.code
start:
    invoke GetModuleHandle, NULL ; отримання дескриптора програми
    mov hInstance,eax ; збереження дескриптора програми
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax
```

```
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
    CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
```

```
mov wc.cbSize,SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
mov wc.lpfWndProc, OFFSET WndProc ; адреса процедури WndProc
mov wc.cbClsExtra,NULL ; кількість байтів для структури
mov wc.cbWndExtra,NULL ; кількість байтів для структури
push hInst ; збереження в стеці дескриптора програми
pop wc.hInstance ; повернення дескриптора в поле структури
mov wc.hbrBackground,COLOR_WINDOW+1 ; колір вікна
mov wc.lpszMenuName,OFFSET MenuName ; ім'я ресурсу меню
mov wc.lpszClassName,OFFSET ClassName ; ім'я класу
invoke LoadIcon,hInstance, IDI_ICON ; ресурс піктограми
mov wc.hIcon,eax ; дескриптор піктограми
mov wc.hIconSm,eax ; дескриптор маленького віконця
invoke LoadCursor,NULL, IDC_ARROW ; ресурс курсора
mov wc.hCursor,eax
```

```
invoke RegisterClassEx, addr wc ; реєстрація класу вікна
```

```
invoke CreateWindowEx, \ ; функція створення вікна за зразком
NULL,ADDR ClassName, \ ; стиль та адреса імені класу
ADDR AppName,\ ; адреса імені вікна
WS_OVERLAPPEDWINDOW, \ ; базовий стиль
CW_USEDEFAULT,CW_USEDEFAULT,\ ; гориз. та верт. координати вікна
700,500, \ ; ширина та висота вікна
0,0, hInst,0 ; дескриптори батьківського вікна, меню, програми
mov hwnd,eax
invoke ShowWindow, hwnd,SW_SHOWNORMAL
invoke UpdateWindow, hwnd
```

```
.WHILE TRUE
```

```
invoke GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення
```

```
.BREAK .IF (!eax)
invoke DispatchMessage, ADDR msg ; відправлення на обслуговування
.ENDW ; закінчення циклу оброблення повідомлень
mov eax,msg.wParam
ret ; повернення з процедури WinMain
WinMain endp ; закінчення процедури з ім'ям WinMain
```

```
WndProc proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,
IParam:LPARAM
LOCAL hdc:HDC ; резервування стека під хендл вікна
.if uMsg==WM_DESTROY
invoke PostQuitMessage,NULL
.ELSEIF uMsg==WM_CREATE ; оброблення повідомлення WM_CREATE
invoke CreatePen,\ ; створення пера
PS_SOLID,40,\ ; завтовшки 40 пікселів
dword ptr color3 ; ясно-бірюзовий
mov hPen1,eah ; збереження отриманого дескриптора пера
.ELSEIF uMsg==WM_COMMAND ; якщо є повідомлення від меню
mov eax,wParam

.if ax==IDM_FUNC
invoke GetDC, hWnd ; отримати дескриптор контексту для функції GDI
mov hdc, eax ; збереження
invoke SelectObject, hdc,hPen1 ; вибрати в контекст нове перо
invoke CreateHatchBrush, \ ; створення кісті для рисування фігури
HS_DIAGCROSS,\ ; стиль – штриховка під 45 градусів зліва та справа
dword ptr color3 ; значення кольору
mov blueBrush,eah ; збереження параметрів щітки
invoke SelectObject, hdc,\ ; вибір об'єкта з параметрами та контекст
blueBrush ; параметри щітки
invoke Ellipse, hdc, \ ; рисування еліпса та дескриптор контексту пристрою
40, 220\ ; x-, y-координати верхнього лівого кута прямокутника
250, 95 ; x-, y-координата нижнього правого кута прямокутника
invoke ReleaseDC, hWnd, hdc
.elseif ax==IDM_ABOUT
invoke MessageBox,0,ADDR About_string,OFFSET AppName,MB_OK
.else
invoke DestroyWindow,hWnd ; знищення вікна
.endif ; закінчення логічної структури
.ELSE
invoke DefWindowProc,hWnd,uMsg,wParam,IParam ; оброблення та
; відправлення повідомлення до функції WndProc
ret
```

```

.ENDIF ; закінчення логічної структури
    xor eax,eax
    ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

**Програма 2.2.** Файл ресурсів:

```

#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_ABOUT 3
#define IDB_MAIN 1
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL "disc.ico"
FirstMenu MENU {
    POPUP "Фигура" {
        MENUITEM "Эллипс",IDM_HELLO
        MENUITEM SEPARATOR
        MENUITEM "Выход",IDM_EXIT
    }
    POPUP "Справка" {
        MENUITEM "About",IDM_ABOUT
    }
}

```

Командний bat-файл має вигляд:

```

ml /c /coff "23_9.asm"
rc "23_9.rc"
link /SUBSYSTEM:windows "23_9.obj" "23_9.res"

```

Результат виконання програми наведено на рис. 2.4.

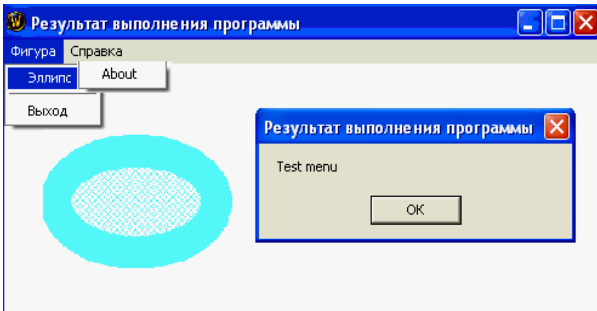


Рис. 2.4. Результат виконання програми з лістингу 2.1



## 2.7. Створення логічних шрифтів

Залежно від принципу зберігання в пам'яті комп'ютера форми символів, розрізняють растрові (точкові) шрифти, які ще називають шрифтами TrueType, та шрифти, що масштабуються. Перевага шрифтів TrueType полягає в тому, що вони дозволяють змінювати в широких межах розмір і інші характеристики символів (наприклад, ширину літер) без зниження якості зображення, що і зумовило їх широке застосування.

Операційна система Windows поставляється з базовим набором растрових шрифтів та шрифтів, що масштабуються, з різноманітними зображеннями і характеристиками (наприклад, є шрифти як з рівною шириною, так і пропорційні).

При розробці власного застосування Windows *на екран можна вивести тільки той шрифт, дескриптор якого завантажений в контекст пристрою*. Для зміни характеристики шрифту треба *створити* новий шрифт із зміненими характеристиками. Це означає, що треба вибрати один зі шрифтів, установлених у системі, і надати йому необхідні характеристики. У системі є декілька готових шрифтів, дескриптори яких можна отримати зі складу, але таким чином можна тільки вибрати шрифт з іншим зображенням символів, змінити ж його характеристики не можна.

Для створення нового шрифту потрібно оголосити в програмі структурну змінну типу LOGFONT, заповнити її поля необхідними значеннями і викликати функцію Windows CreateFontIndirect. Ця функція поверне дескриптор нового шрифту; після вибору отриманого дескриптора в контекст пристрою будь-яка функція виведення на екран тексту використовуватиме саме цей шрифт. Якщо в застосунку бажано використовувати різні шрифти, їх можна створити заздалегідь і вибрати в контекст пристрою в міру необхідності.

Неправильне установлення членів структури LOGFONT призводить до того, що Windows добирає найбільш відповідний шрифт.

Структура LOGFONT має такий склад параметрів:

```
LOGFONT struc
IfHeight dw ? ; висота
IfWidth dw ? ; середня ширина
IfEscapement dw ? ; кут нахилу в 1/10 градуса
IfOrientation dw ? ; не використовується
IfWeight dw ? ; жирність: FW_NORMAL, FW_BOLD, ; FW_LIGHT та ін.
IfItalic db ? ; якщо = 1, то курсив
IfUnderline db ? ; якщо = 1, то підкреслення
```

```

IfStrikeOut db ? ; якщо = 1, то перекреслювання
IfCharSet db ? ; набір символів; зазвичай = 0
lfOutPrecision db ? ; необхідна точність відповідності; звичайно = 0
  IfClipPrecision db ? ; спосіб вирізки частини символу; зазвичай = 0
IfQuality db ? ; якість, зазвичай = 0
IfPitchAndFamily db ? ; крок і сімейство, зазвичай = 0
IfFaceName db Lf_facesize dup (?) ; ім'я зображення шрифту
  LOGFONT ends

```

Найбільш важливим є останній параметр цієї структури – символний масив lf.FaceName. У нього треба скопіювати повне ім'я шрифту з числа шрифтів, установлених у Windows, наприклад, Times New Roman Cyr, Arial Cyr, Courier New Cyr й так далі. У параметр lf.FaceName заноситься не адреса імені шрифту, а сам символний рядок з ім'ям. При цьому використовується команда movsb.

Приклад:

```

.data
Hello db "Трикутник",0 ; текст, який виводиться
myFont LOGFONT <20, 20, 300, 0, FW_BOLD,1,1,0,0,0,0,0,0,0,0,0,0,0,'Times New Roman'> ; лог. шрифт
color2 RGBQUAD <0,165,190> ; колір
hfont dd 0
...

.ELSEIF uMsg==WM_PAINT ; якщо рисування
invoke SelectObject,hdc,hfont ; вибір у контекст шрифту
invoke GetClientRect,hWnd,ADDR rect ; занесення в структуру rect
; характеристик вікна
invoke SetTextColor,hdc,addr color2 ; встановлення кольорового тексту
invoke SetBkMode,hdc,00000000h ; встановлення фону за текстом
invoke TextOut,hdc,300,150,addr Hello,9

```

Природно, що повідомлення можливо й перевернути на 180°, наприклад:

```

myLog LOGFONT <24,20,0,0,FW_BOLD,1,1,0,0,0,0,0,0,0,0,0,'Times New Roman'>
myLog2 LOGFONT <24,20,1800,0,FW_BOLD,1,1,0,0,0,0,0,0,0,0,0,'Times New Roman'>

```

Але в такому випадку повідомлення відобразиться ще й дзеркально (рис. 2.5).



Рис. 2.5. Надпис, який перевернутий на 180°

## 2.8. Виведення тексту

Win32 GDI містить функції для форматування і рисування тексту в клієнтській ділянці вікна або на паперовій сторінці принтера.

Функція `TextOut` виводить текстовий рядок і має прототип:

```
TextOut,  
HDC hdc,                ; дескриптор контексту пристрою  
int nXStart,            ; x-координата стартової позиції  
int nYStart,            ; y-координата стартової позиції  
LPCTSTR lpString,      ; указівник на символьний рядок  
int cbString            ; число символів у рядку
```

Функція забезпечує виведення рядка за адресою `lpString`, розміщуючи текст у заданій позиції з урахуванням поточного режиму вирівнювання. Кількість символів, що виводяться, задається параметром `cbString`.

Поточний режим **вирівнювання тексту** реалізований у вигляді опорної точки. Це значення можна змінити за допомогою функції `SetTextAlign`, передаючи другому параметру бітову маску, утворену об'єднанням відповідних прапорців:

```
SetTextAlign(HDC hdc. UINT fMode)
```

Для забезпечення роботи з **табульованим текстом** Windows використовує функції `TabbedTextOut` та `GetTabbedTextExtent`.

Функція `TabbedTextOut` має такий прототип:

```
LONG TabbedTextOut(HDC hdc, int X, int Y, LPCTSTR lpString, \  
int nCount, int iNumTabs, CONST LPINT lpnTabStops, \  
int xTabOrigin);
```

Перші п'ять параметрів мають те ж значення, що й функція `TextOut`. Шостий параметр, `iNumTabs`, визначає кількість позицій табуляції. Сьомий параметр, `lpnTabStops`, містить указівник на масив позицій табуляції, заданих у логічних одиницях. Позиції табуляції повинні бути відсортовані в зростаючому порядку.

Якщо шостий параметр дорівнює нулю та одночасно сьомий параметр дорівнює `NULL`, то позиції табуляції встановлюються через однакові проміжки (восьмиризова середня ширина символів). Якщо шостий

параметр дорівнює одиниці, то перший елемент масиву `lpnTabStops` містить число символівних позицій, яке кожного разу додається для визначення наступної позиції табуляції.

Останній параметр `xTabOrigin` задає логічну координату за горизонталлю *точки рахування* позицій табуляції.

**Міжсимвольний інтервал** додається до кожного символу, включаючи символи пропуску, коли GDI виводить рядок тексту. За умовчанням цей атрибут дорівнює нулю. Функція `SetTextCharacterExtra` присвоює йому нове цілочислове значення в логічних одиницях, повертаючи попереднє значення. Функція `GetTextCharacterExtra` повертає поточне значення міжсимвольного інтервалу.

Для **виведення тексту** в межах використовуються функції `DrawText` та `DrawTextEx`.

Функція `DrawText` має такий прототип:

```
DrawText,
HDC hDC,           ; дескриптор контексту пристрою
LPCTSTR lpString, ; указівник на текстовий рядок
int nCount,        ; довжина тексту
LPRECT lpRect,    ; прямокутник, в якому розміщується текст
UINT uFormat      ; прапорці форматування;
```

Функція `DrawText` виводить текстовий рядок у прямокутній ділянці, заданій параметром `lpRect`. Функція `DrawText` приймає як параметри вказівник на символівний рядок та довжину рядка. Якщо параметру `nCount` присвоїти значення `-1`, то функція сама визначить довжину рядка. Якщо ж рядок не має нуля на кінці, то необхідно вказати його довжину в аргументі `nCount`. Параметр `uFormat` визначає метод форматування тексту. Його значення є бітовою маскою, утвореною об'єднанням прапорців форматування. Найбільш важливі прапорці, що часто вживаються, наведено в табл. 2.11.

Таблиця 2.11 – Прапорці форматування для функції `DrawText`

Прапорці	Значення	Опис
<code>DT_LEFT</code>	<code>0x0000</code>	Вирівнювання тексту вліво
<code>DT_CENTER</code>	<code>0x0001</code>	Центрування за горизонталлю
<code>DT_RIGHT</code>	<code>0x0002</code>	Вирівнювання тексту вправо
<code>DT_TOP</code>	<code>0x0000</code>	Початок розміщення – у верхній частині прямокутника

Продовження табл. 2.11

DT_CENTER	0x0004	Центрування за вертикаллю (використовується тільки разом з DT_SINGLELINE)
DT_BOTTOM	0x0008	Розміщення внизу прямокутника (використовується тільки разом з DT_SINGLELINE)
DT_WORD-BREAK	0x0010	Перенесення тексту на наступний рядок після закінчення чергового слова, якщо частина тексту, що залишилася, не поміщається за шириною прямокутника виведення. Прапорець працює, тільки якщо не вказаний прапорець DT_SINGLELINE
DT_SINGLE-LINE	0x0020	Виведення тексту тільки в один рядок. Символи повернення каретки та перекладу рядка не сприймаються як керуючі символи (виводяться як • або Q)
DT_EXPANDTABS	0x0040	Інтерпретація символів \t, які задають табуляцію. За умовчанням крок табуляції дорівнює восьмиразовій середній ширині символів
DT_NOPREFIX	0x0800	Відміння оброблення префіксів &. Зазвичай DrawText інтерпретує префікс & як директиву «підкреслити наступний за префіксом символ», а послідовність && – як директиву виведення одиничного символу &. З прапорцем DT_NOPREFTX символ & сприймається нарівні з іншими символами, але не як префікс
DT_PATH_ELLIPSIS	0x4000	Якщо рядок не поміщається за шириною прямокутника, частина символів в середині рядка заміщається багатоточкою (використовується тільки разом з DT_SINGLELINE)
DT_END_ELLIPSIS	0x8000	Якщо рядок не поміщається за шириною прямокутника, частина символів у кінці рядка заміщається трьома точками (використовується тільки разом з DT_SINGLELINE)
DT_CALCRECT		Визначає ширину і висоту прямокутника. Якщо використовуються багаторазові лінії тексту, то DrawText використовує ширину прямокутника, вказаного lpRect параметром. DrawText повертає висоту тексту

Закінчення табл. 2.11

DT_EDITCONTR OL		Дублює характеристики багаторядкового елемента редагування тексту
DT_EXTER- NALLEADING		Включення зовнішнього шрифту
DT_MOD- IFYSTRING		Змінює рядок тексту. Діє для прапорців DT_END_ELLIPSIS, DT_PATH_ELLIPSIS
DT_NOCLIP		Розташування без відсікання
DT_RTL- READING		Розташування справа наліво для єврейського або арабського шрифтів
DT_TABSTOP		Встановлює позиції табуляції. Типове число символів за таблицю – вісім
DT_VCENTER		Зосереджує текст вертикально

Функція DrawText виводить 488 знаків з пропусками та службовими символами.

Функція ExtTextOut, що є розширеною версією функції TextOut, має такий прототип:

```
ExtTextOut(HDC hdc, int X, int Y, \
    UINT fuOptions, CONST RECT* lprc, \
    LPCTSTR lpString, UINT cbCount, \
    CONST INT* lpDx);
```

Для створення кольорового фону використовується функція CreateSolidBrush, параметром якої в даному випадку є число **00A5FF30h**, яке характеризує колір фону. Це число може бути отримане різними шляхами. У цьому випадку воно отримане з програми Paint.net з вікна Параметри\Больше. Можна також використати й відповідну кольорову структуру.

При створенні логічного шрифту було використано 13 параметрів, а чотирнадцятий параметр не був використаний.

Для обведення будь-якої фігури (**рисуння бордюрів**) використовується функція DrawEdge. Її опис такий:

```
DrawEdge, \ ; рисуння бордюру
hdc, \ ; хендл пристрою
var grc: TRect \ ; указівник на координати прямокутника
edge: UINT \ ; вигляд внутрішнього та зовнішнього краю
grfFlags: UINT ; вигляд бордюру
```

Рамка рисується у вигляді комбінації з двох прямокутників – внутрішнього (inner) і зовнішнього (outer). Кожен з них може бути опуклим (raised) або угнутим (sunken). Тип рамки визначається параметром `edge`.

Структура `Trect` визначає логічні координати прямокутника бордюру.

Параметр `edge` вказує тип рисування внутрішнього і зовнішнього країв бордюру. Параметр повинен бути комбінацією одного прапорця внутрішньої межі і одного прапорця зовнішньої межі.

Прапорці внутрішньої межі:

`BDR_RAISEDINNER` – опуклий;

`BDR_SUNKENINNER` – угнутий.

Прапорці зовнішньої межі:

`BDR_RAISEDOUTER` – опуклий;

`BDR_SUNKENOUTER` – угнутий.

Крім того, прапорець межі може набувати ще декількох значень:

`EDGE_BUMP` – комбінація з прапорців `BDR_RAISEDOUTER` та `BDR_SUNKENINNER`;

`EDGE_ETCHED` – комбінація з прапорців `BDR_SUNKENOUTER` та `BDR_RAISEDINNER`;

`EDGE_RAISED` – комбінація з прапорців `BDR_RAISEDOUTER` та `BDR_RAISEDINNER`;

`EDGE_SUNKEN` – комбінація з прапорців `BDR_SUNKENOUTER` та `BDR_SUNKENINNER`.

Параметр `grfFlags`: визначає рисування бордюру та його типу:

`BF_BOTTOM` – рисується нижня смуга бордюру;

`BF_TOP` – рисується верхня смуга бордюру;

`BF_LEFT` – рисується ліва смуга бордюру;

`BF_RIGHT` – рисується права смуга бордюру;

`BF_BOTTOMLEFT` – рисується лівий нижній кут;

`BF_BOTTOMRIGHT` – рисується лівий правий кут;

`BF_TOPLEFT` – рисується лівий верхній кут;

`BF_TOPRIGHT` – рисується правий верхній кут;

`BF_RECT` – рисується вся рамка;

`BF_DIAGONAL` – рисує межу за діагоналлю. Прапорці цієї групи частіше використовуються для рисування кнопок, розбитих за діагоналлю на дві секції;

BF\_DIAGONAL\_ENDBOTTOMLEFT – діагональ виходить з верхнього правого кута, заходить в нижній лівий;  
 BF\_DIAGONAL\_ENDBOTTOMRIGHT – діагональ виходить з верхнього лівого кута, заходить в нижній правий;  
 BF\_DIAGONAL\_ENDTOPLEFT – діагональ виходить з нижнього правого кута, заходить у верхній лівий;  
 BF\_DIAGONAL\_ENDTOPRIGHT – діагональ виходить з нижнього лівого кута, заходить у верхній правий;  
 BF\_FLAT – плоска межа (стиль OfficeXP);  
 BF\_SOFT – м'які кнопки (стиль OfficeXP);  
 BF\_MONO – рисує одновимірну рамку;  
 BF\_MIDDLE – заливає внутрішню ділянку рамки кольором поточної щітки;  
 BF\_ADJUST – коректує параметр `qrc` так, що після обрисовування він відповідає внутрішній ділянці рамки. Зручно застосовувати для подальшого рисування надалі.

Програму рисування кольорового трикутника зі зміною фону вікна, фону заливки трикутника з виведенням у вікно кольорового тексту та створення заливки фону тексту наведено в програмі 2.3.

### Програма 2.3:

```

title Rysovaniy A.N.
; рисування кольорового трикутника + фону вікна + фону заливки +
; + фону трикутника + кольорового шрифту + фону заливки шрифту
.686          ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\include\windows.inc
include \masm32\macros\macros.asm
uselib user32, kernel32, gdi32
  
```

**WinMain proto** hInst:HINSTANCE, CmdShow:DWORD ; прототип процедури

```

.data          ; директива визначення даних
  ClassName db "Rysovaniy", 0
  AppName   db "Rysovaniy A.N.", 0
  Hello     db "Трикутник", 0          ; текст, який виводиться
  pts POINT <350,550>,<450,300>,<550,550>
  myFont LOGFONT <30, 30, 685, 0, FW_BOLD, 1, 1, 0, 0, 0, 0, 0, \
    'Times New Roman'> ; логічний шрифт
  Edge RECT <200,250,600,600> ; координати бордюру
  hPen1 dd 0          ; комірка збереження дескриптора пера
  
```



```

hBrush dd 0 ; комірка збереження дескриптора шрифту
hfont dd 0
color1 RGBQUAD <255,244,40> ; колір пера трикутника
color2 RGBQUAD <0,165,190> ; колір заливки трикутника
.data? ; директива невизначених даних
hInstance HINSTANCE ?

.code ; директива початку сегмента команд
start: ; мітка початку програми з ім'ям start
invoke GetModuleHandle, NULL ; отримання дескриптора програми
mov hInstance, eax ; збереження дескриптора програми
invoke WinMain, hInstance, SW_SHOWDEFAULT
invoke ExitProcess, eax ; повернення управління ОС та вивільнення ресурсів

```

### **WinMain proc** hInst:HINSTANCE, CmdShow:DWORD

```

LOCAL wc:WNDCLASSEX ; резервування стека під структуру
LOCAL msg:MSG ; резервування стека під структуру MSG
LOCAL hwnd:HWND ; резервування стека під хендл програми
push hInstance ; збереження в стеку дескриптора програми
pop wc.hInstance ; повернення дескриптора в поле структури
mov wc.cbSize, SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
mov wc.lpszWndProc, OFFSET WndProc ; адреса процедури WndProc
mov wc.hbrBackground, COLOR_WINDOW+1 ; колір вікна
invoke GetStockObject, WHITE_BRUSH ; читання описувача
invoke CreateSolidBrush, 00FF96F0h ; колір фону вікна
mov wc.hbrBackground, eax ; колір заповнення вікна
mov wc.lpszMenuName, NULL ; ім'я ресурсу меню
mov wc.lpszClassName, OFFSET ClassName ; ім'я класу
invoke LoadIcon, NULL, IDI_APPLICATION ; ресурс піктограми
mov wc.hIcon, eax ; дескриптор «великої» піктограми
mov wc.hIconSm, eax ; дескриптор маленького віконця
invoke LoadCursor, NULL, IDC_ARROW ; ресурс курсора
mov wc.hCursor, eax
mov wc.cbClsExtra, NULL ; кількість байтів для структури
mov wc.cbWndExtra, NULL ; кількість байтів для додаткових структур
invoke RegisterClassEx, ADDR wc ; функція реєстрації класу вікна
invoke CreateWindowEx, 0, ADDR ClassName, \ ; стиль та адреса імені класу
ADDR AppName, \ ; адреса імені вікна
WS_OVERLAPPEDWINDOW, \ ; базовий стиль
CW_USEDEFAULT, CW_USEDEFAULT, \ ; гориз. та верт. координати вікна
CW_USEDEFAULT, CW_USEDEFAULT, \ ; ширина та висота вікна
NULL, NULL, hInst, NULL ; дескриптор програми
mov hwnd, eax
invoke ShowWindow, hwnd, SW_SHOWNORMAL ; видимість вікна

```

```

.WHILE TRUE ; поки істинне, то
    invoke GetMessage, ADDR msg, NULL, 0, 0 ; читання повідомлення
    or eax, eax ; формування ознак
    jz Quit ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправка до WndProc proc
.ENDW
Quit:
    mov eax, msg.wParam
    ret ; повернення з процедури
WinMain endp ; закінчення процедури з ім'ям WinMain

```

```

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM
LOCAL hdc:HDC ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
PAINTSTRUCT
LOCAL rect:RECT ; резервування стека під структуру RECT
.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
    invoke DeleteObject, hPen1 ; видаляє Handle з пам'яті і звільняє ресурси
    invoke DeleteObject, hFont ; видаляє Handle з пам'яті і звільняє ресурси
    invoke DeleteObject, hBrush ; видаляє Handle з пам'яті і звільняє ресурси
    invoke PostQuitMessage, NULL ; передача повідомлення про знищення
.ELSEIF uMsg==WM_CREATE ; якщо є повідомлення про створення вікна
    invoke CreatePen, PS_SOLID, 40, dword ptr color1 ; створення щітки
    mov hPen1, eax ; збереження дескриптора пера
    invoke CreateFontIndirect, ADDR myFont ; створення логічного шрифту
    mov hFont, eax ; збереження дескриптора шрифту
    invoke CreateSolidBrush, 005B76FFh ; створення щітки для кольору фону
    ; заливки трикутника
    mov hBrush, eax
.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
    invoke BeginPaint, hWnd, ADDR ps ; виклик проц. та заповнення структури
    mov hdc, eax ; збереження контексту
    invoke SelectObject, hdc, hFont ; вибір в контекст шрифту
    invoke GetClientRect, hWnd, ADDR rect ; занесення в структуру rect
    ; характеристик вікна
    invoke SetTextColor, hdc, addr color2 ; установлення кольорового тексту
    invoke SetBkMode, hdc, 00000000h ; установлення фону за текстом
    invoke TextOut, hdc, 250, 580, \
    addr Hello, 9 ; виведення тексту, координати
    ; адреса зберігання тексту та кількість байтів тексту
    invoke SelectObject, hdc, hPen1 ; вибір у контекст пера
    invoke SelectObject, hdc, hBrush ; вибір у контекст щітки
    invoke DrawEdge, hdc, ADDR Edge, EDGE_SUNKEN, BF_RECT ; рамка
    invoke Polygon, hdc, ADDR pts, 3 ; рисування трикутника
    invoke EndPaint, hWnd, ADDR ps ; закінчення рисування
.ELSE ; інакше

```

```

invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; оброблення та
; відправлення повідомлення до функції WndProc
ret ; повернення з процедури
.ENDIF ; закінчення логічної структури .IF – .ELSEIF
xor eax,eax ; підготування до закінчення
ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

На рис. 2.6 наведено результат виконання програми 2.3.

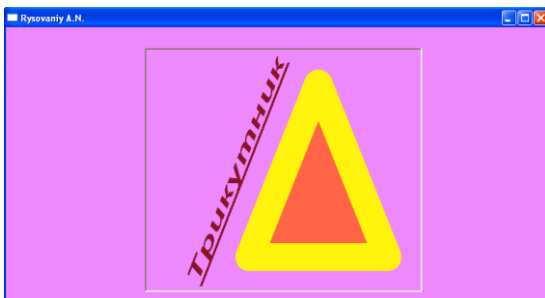


Рис. 2.6. Результат виконання програми 2.3

Для зарисовування внутрішньої частини об'єкта також використовують функцію пріоритетного змішування кольору щітки або пера з існуючим зображенням SetROP2 з такими параметрами:

```

invoke SetROP2,\ ; пріоритетне змішування кольорів
hdc\ ; ідентифікатор контексту пристрою
integer ; ціле число або ідентифікатор

```

Ідентифікатор конкретизує новий метод змішування. Цей параметр може бути одним з таких значень:

- R2\_BLACK – піксель завжди становить 0;
- R2\_COPYPEN – колір пера;
- R2\_MASKNOTPEN – комбінація кольорів як екрана, так і протилежностей пера;
- R2\_MASKPEN – комбінація кольорів пера та екрана;
- R2\_MASKPENN0T – комбінація кольорів пера й протилежності екрана;
- R2\_MERGENOTPEN – комбінація кольору екрана і протилежності кольору пера;

R2\_MERGEPEEN – комбінація кольору пера та кольору екрана;  
 R2\_MERGEPEENNOT – комбінація кольору пера та протилежності кольору екрана;  
 R2\_NOP – залишається незмінним;  
 R2\_NOT – протилежність кольору екрана;  
 R2\_NOTCOPYPEN – протилежність кольору пера;  
 R2\_NOTMASKPEN – протилежність кольору R2\_MASKPEN;  
 R2\_NOTMERGEPEN – протилежність кольору R2\_MERGEPEEN;  
 R2\_NOTXORPEN – протилежність кольору R2\_XORPEN;  
 R2\_WHITE – завжди складає 1;  
 R2\_XORPEN – комбінація кольорів пера й екрана.

## 2.9. Спіраль Архімеда з плавною зміною кольору фігури

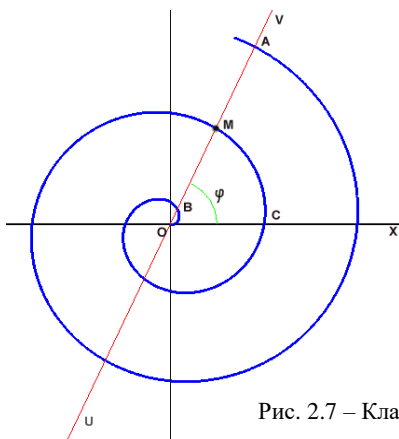


Рис. 2.7 – Класична спіраль Архімеда

Архімедова спіраль ([http://ru.wikipedia.org/wiki/Архимедова\\_спираль](http://ru.wikipedia.org/wiki/Архимедова_спираль)) – спіраль, плоска крива, траєкторія точки М (рис. 2.7) яка рівномірно рухається уподовж лінії OV з початком у точці O, тоді як сама лінія OV рівномірно обертається навколо точки O.

Іншими словами, відстань  $\rho = OM$  пропорційна куту повороту  $\varphi$  лінії OV. Повороту лінії OV на один і той же кут відповідає один і той же приріст  $\rho$ :

$$\rho = k\varphi,$$

де  $k$  – зсув точки М по лінії  $r$ , при повороті на кут, який дорівнює одному радіану (<http://rusproject.narod.ru/article/polar.htm>).

Повороту прямої на  $2\pi$  відповідає зсув  $a = |BM| = |MA| = 2k\pi$ . Число  $a$  — називається кроком спіралі. Рівняння Архімедової спіралі у полярній системі координат можна переписати так:  $P = (a/2\pi)\varphi$ .

Полярна система координат – двовимірна система координат, в якій кожна точка на площині визначається двома числами – полярним кутом і полярним радіусом.

При обертанні лінії проти годинникової стрілки виходить права спіраль, при обертанні за годинниковою стрілкою – ліва спіраль. Позитивним значенням відповідає права спіраль, негативним – ліва спіраль. Якщо точка  $M$  рухатиметься по прямої  $UV$  з негативних значень через центр обертання  $O$  і далі в позитивні значення упродовж прямій  $UV$ , то точка  $M$  опише обидві спіралі.

Відстані між точками  $B$  та  $M$ ,  $M$  та  $A$  дорівнюють кроку спіралі  $2\kappa$ . При розкручуванні спіралі відстань від точки  $O$  до точки  $M$  прямує до нескінченності. При цьому крок спіралі залишається постійним (кінцевим), тобто чим далі від центра, тим ближче витки спіралі за формою наближаються до кола.

Простіше використовувати (<http://hijos.ru/2011/03/09/archimedova-spiral/>) параметризацію спіралі (декартова система координат) за формулами:

$$\begin{aligned}x &= \varphi \cos \varphi; \\y &= \varphi \sin \varphi.\end{aligned}$$

У декартовій або прямокутній системі координат відносини між координатами можна встановити тільки шляхом застосування тригонометричних рівнянь.

У програмі 2.4 наведено файл ресурсів програми.

#### **Програма 2.4.** Файл ресурсів програми:

```
#define IDM_INFO 1
#define IDM_USL 3
#define IDM_EXIT 2
#define IDM_M1 11
#define IDM_M2 12
#define IDM_M3 13
#define IDM_V1 21
#define IDM_V2 22
#define IDM_V3 23
#define IDM_V4 24
#define IDM_V5 25
#define IDM_V6 26
#define IDM_V7 27
#define IDM_V8 28
#define IDM_V9 29
#define IDM_V0 20
#define IDL_ICON 1001
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL "Spiral1.ico"
```

```

FirstMenu MENU {
    POPUP "Программа"{
        MENUITEM "Условие...", IDM_USL
        MENUITEM "Об авторе...", IDM_INFO
        MENUITEM SEPARATOR
        MENUITEM "Выход", IDM_EXIT
    }
    POPUP "Масштаб"{
        MENUITEM "1:2", IDM_M1
        MENUITEM "1:1", IDM_M2
        MENUITEM "2:1", IDM_M3
    }
    POPUP "Количество витков"{
        MENUITEM "1", IDM_V1
        MENUITEM "2", IDM_V2
        MENUITEM "3", IDM_V3
        MENUITEM "4", IDM_V4
        MENUITEM "5", IDM_V5
        MENUITEM "6", IDM_V6
        MENUITEM "7", IDM_V7
        MENUITEM "8", IDM_V8
        MENUITEM "9", IDM_V9
        MENUITEM "10", IDM_V0
    }
}
}

```

Програма виконання прикладу 2.1 наведена в програмі 2.5.

**Програма 2.5.** Рисування спіралі Архімеда з плавною зміною кольору фігури:

```

.686 ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер

```

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD ; прототип процедури

```

include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
uselib user32, kernel32, fpu, gdi32

```

**@ MACRO b0,b1,b2,b3**

**b0**

**b1**

**b2**

**b3**

**ENDM**

```

.data          ; директива визначення даних
  ClassName db "Class", 0
  AppName db "Спираль Архимеда",0
  MenuName db "FirstMenu",0
  info_string db "Автор: ",0
  usl_string db "Построение спирали Архимеда",0dh,0ah,
    "с количеством витков от 1 до 10",0dh,0ah,
    "в трех вариантах масштаба",0
  info_caption db "Об авторе",0
  usl_caption db "Условие",0
  mas dd 07AB7h          ; кількість циклів
  two dd 2
  alpha dd 0.0          ; кутова координата
  delta dd 0.001        ; збільшення координати
  xdiv2 dd ?            ; середина по X та Y
  ydiv2 dd ?
  tmp dd 0              ; тимчасова змінна
  K1 dd 2.5             ; масштабні коефіцієнти
  K2 dd 5.0
  K3 dd 10.0
  divK dd 5.0
  xr dd 0               ; координати функції
  yr dd 0
  colour dd 32000      ; початкове значення кольору (зелений)
  count db 0

```

```

.data?        ; директива невизначених даних
  hInstance HINSTANCE ?
  CommandLine LPSTR ?

```

```

.const        ; визначення констант
  IDM_INFO equ 1
  IDM_USL equ 3
  IDM_EXIT equ 2
  IDM_M1 equ 11          ; кількість витків спіралі
  IDM_M2 equ 12
  IDM_M3 equ 13
  IDM_V1 equ 21
  IDM_V2 equ 22
  IDM_V3 equ 23
  IDM_V4 equ 24
  IDM_V5 equ 25
  IDM_V6 equ 26
  IDM_V7 equ 27
  IDM_V8 equ 28

```

```
IDM_V9 equ 29
IDM_V0 equ 20
IDI_ICON equ 1001
```

```
.code ; директива початку сегмента команд
start: ; мітка початку програми з ім'ям start
    invoke GetModuleHandle,NULL ; отримання дескриптора програми
    mov hInstance,eax ; збереження дескриптора програми
    invoke GetCommandLine
    mov CommandLine,eax
invoke WinMain,hInstance,NULL,CommandLine,SW_SHOWDEFAULT
invoke ExitProcess,eax ; повернення управління ОС Windows та
; визволення ресурсів
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
    CmdShow:DWORD
    LOCAL wc:WNDCLASSEX ; резервування стека під структуру
    LOCAL msg:MSG ; резервування стека під структуру MSG
    LOCAL hwnd:HWND ; резервування стека під хендл програми
    @<mov wc.cbSize,SIZEOF WNDCLASSEX>,<mov wc.cbClsExtra,0>
    @<mov wc.style,CS_HREDRAW or CS_VREDRAW >,<mov wc.cbWndEx-
    tra,0>
    @<mov wc.lpfWndProc,OFFSET WndProc>,<push hInst>,<pop wc.hIn-
    stance>
    @<mov wc.hbrBackground,COLOR_WINDOW+1>,<mov wc.lpszMenuName,OFFSET
    MenuName>
    mov wc.lpszClassName,OFFSET ClassName ; ім'я класу
    invoke LoadIcon,hInstance,IDI_ICON ; ресурс піктограми
    @< mov wc.hIcon,eax >,< mov wc.hIconSm,eax >
    invoke LoadCursor,NULL,IDC_ARROW ; ресурс
    mov wc.hCursor,eax ; дескриптор курсора
    invoke RegisterClassEx,addr wc ; реєстрація класу вікна
    INVOKE CreateWindowEx,\ ; функція створення вікна за зразком
    NULL,ADDR ClassName,\ ; стиль та адреса імені класу
    ADDR AppName,\ ; адреса імені вікна
    WS_OVERLAPPEDWINDOW,\ ; базовий стиль
    200,200,600,600\ ; x0, Y0, Δx, Δy координати вікна
    0,0,hInst,0 ; дескриптори батьківського вікна, меню, програми
    mov hwnd,eax
    INVOKE ShowWindow, hwnd,SW_SHOWNORMAL ; видимість вікна
    INVOKE UpdateWindow, hwnd
    .WHILE TRUE ; поки істинне, то
    invoke GetMessage, ADDR msg, NULL, 0, 0 ; читання повідомлення
    or eax, eax ; формування ознак
    jz Quit ; перейти на мітку Quit, якщо eax = 0
    invoke DispatchMessage, ADDR msg ; відправлення на обслуговування
    .ENDW ; закінчення циклу оброблення повідомлень
```



Quit:

```
    mov  eax,msg.wParam
    ret      ; повернення з процедури WinMain
WinMain endp ; закінчення процедури з ім'ям WinMain
```

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

```
    LOCAL rect:RECT      ; резервування стека під структуру RECT
    LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
    LOCAL hdc:HDC        ; резервування стека під хендл вікна
```

**.IF uMsg==WM\_DESTROY** ; якщо є повідомлення про знищення вікна  
 invoke PostQuitMessage,NULL ; передача повідомлення  
WM\_Quit

**.ELSEIF uMsg==WM\_COMMAND** ; якщо є повідомлення з меню

```
    mov  eax,wParam ; формування ознак повідомлення з меню
    .IF ax==IDM_INFO ; виведення інформації про автора
    invoke MessageBox,NULL,ADDR info_string,ADDR info_caption,MB_OK
    .ELSEIF ax==IDM_USL ; виведення умови
    invoke MessageBox,NULL,ADDR usl_string,ADDR usl_caption,MB_OK
    .ELSEIF ax==IDM_M1 ; визначення коефіцієнта масштабу
```

```
@< mov  ebx, K1 ; divK=2.5>,< mov  divK, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_M2
```

```
@< mov  ebx, K2 ; divK=5.0>,< mov  divK, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_M3
```

```
    mov  ebx, K3 ; divK=10.0
    mov  divK, ebx
```

```
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_V1 ; визначення кількості циклів
```

```
@< mov  ebx, 0188Bh ; mas=1*Pi*2000>,< mov  mas, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_V2
```

```
@< mov  ebx, 03116h ; mas=2*Pi*2000>,< mov  mas, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_V3
```

```
@< mov  ebx, 049A1h ; mas=3*Pi*2000>,< mov  mas, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_V4
```

```
@< mov  ebx, 0622Ch ; mas=4*Pi*2000>,< mov  mas, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
    .ELSEIF ax==IDM_V5
```

```
@< mov  ebx, 07AB7h ; mas=5*Pi*2000>,< mov  mas, ebx >
    invoke InvalidateRect, hWnd, NULL, TRUE
```

```

.ELSEIF ax==IDM_V6
@< mov ebx, 09343h ; mas=6*Pi*2000>,< mov mas, ebx >
  invoke InvalidateRect, hWnd, NULL, TRUE
.ELSEIF ax==IDM_V7
@< mov ebx, 0ABCEh ; mas=7*Pi*2000>,< mov mas, ebx >
  invoke InvalidateRect, hWnd, NULL, TRUE
.ELSEIF ax==IDM_V8
@< mov ebx, 0C459h ; mas=8*Pi*2000>,< mov mas, ebx >
  invoke InvalidateRect, hWnd, NULL, TRUE
.ELSEIF ax==IDM_V9
@< mov ebx, 0DCE4h ; mas=9*Pi*2000>,< mov mas, ebx >
  invoke InvalidateRect, hWnd, NULL, TRUE
.ELSEIF ax==IDM_V0
@< mov ebx, 0F56Fh ; mas=10*Pi*2000>,< mov mas, ebx >
  invoke InvalidateRect, hWnd, NULL, TRUE
.ELSE
  invoke DestroyWindow,hWnd      ; знищення вікна
.ENDIF

```

**.ELSEIF uMsg==WM\_PAINT** ; якщо є повідомлення про

перерисовування

invoke BeginPaint, hWnd, addr ps ; виклик підготовчої процедури

; та заповнення структури

mov hdc, eax ; збереження контексту

invoke GetClientRect, hWnd, addr rect; занесення в структуру rect харак-  
теристик вікна

mov alpha, 0 ; скидання кутової координати

finit ; ініціювання співпроцесора

fild rect.right ; st(0) = (ширина вікна)

fild two ; st(0) = 2

fdivp st(1), st ; st(0) = (ширина вікна)/2

fistp dword ptr xdiv2 ; збереження середини за X

fild rect.bottom ; st(0) = (висота вікна)

fild two ; st(0) = 2

fdivp st(1), st ; st(0) = (висота вікна)/2

fistp dword ptr ydiv2 ; збереження середини за Y

invoke MoveToEx, hdc, xdiv2, ydiv2, NULL ; переміщення точки початку

; рисування у середину вікна

mov ecx, mas ; визначення кількості циклів

push ecx ; збереження в стеку кількості циклів

I1: finit ; ініціювання співпроцесора

fild alpha ; st(0) = alpha

fcos ; st(0) = cos(alpha)

fild alpha ; st(0) = alpha

fmul divK ; st(0) = alpha \* divK, st(1) = cos(alpha)

fmul ; st(0) = st(0) \* st(1) = alpha \* divK \* cos(alpha)

```

frndint          ; округлення st(0)
fild xdiv2
fadd             ; додання середини по X
fistp dword ptr xr ; збереження X у змінну для виведення на екран
fld alpha       ; st(0) = alpha
fsin            ; st(0) = sin(alpha)
fld alpha       ; st(0) = alpha
fmul divK      ; st(0) = alpha * divK, st(1) = sin(alpha)
fmul            ; st(0) = st(0) * st(1) = alpha * divK * cos(alpha)
frndint        ; округлення st(0)
fstp tmp
fild ydiv2
fsub tmp        ; віднімання від середини за Y
fistp dword ptr yr ; збереження X у змінну для виведення на екран
invoke SetPixel, hdc, xr, yr, colour ; рисування
    .IF count==2          ; зміна кольору кожні X пікселів
        @<inc colour>,<mov count,0>
    .ELSE
        inc count
    .ENDIF
    fld delta ; збільшення кутової координати
    fld alpha
    fadd
    fstp alpha
    pop ecx ; повернення зі стека кількості циклів
    dec ecx ; зменшення лічильника
    push ecx
    jz l2 ; продовження рисування
    jmp l1 ; вихід із циклу
l2:    pop ecx
        invoke EndPaint, hWnd, addr ps ; закінчення рисування
        mov eax, 0

.ELSE ; інакше
invoke DefWindowProc, hWnd, uMsg, wParam, lParam ; оброблення
        ; та відповідь повідомлення до функції WndProc
    ret
.ENDIF ; закінчення логічної структури .IF - .ELSE
    xor eax, eax
    ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

**Програма 2.6.** Командний bat-файл для створення виконавчого файлу:

```

del spiral1.exe
ml /c /coff "spiral1.asm"
rc "spiral1.rc"
link /SUBSYSTEM:windows "spiral1.obj" "spiral1.res"
del spiral1.obj
del spiral1.res
pause
start spiral1.exe

```

Результат виконання програми наведено на рис. 2.8.

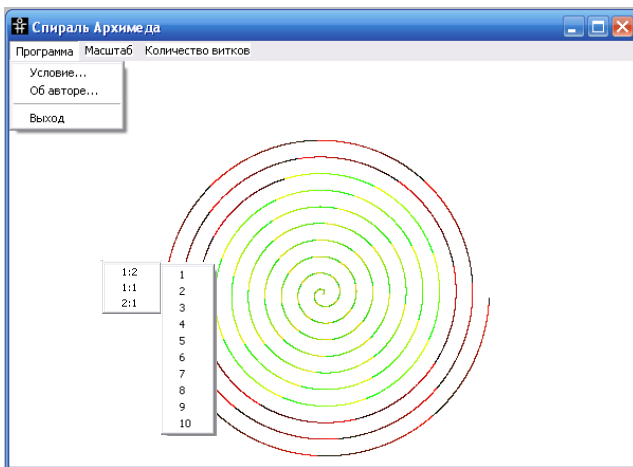


Рис. 2.8. Результат виконання програми 2.5

Функція рисування точки `SetPixel` установлює колір точки з заданими координатами:

```

COLORREF WINAPI SetPixel(
    HDC hdc,          // контекст відображення
    int nXPos,       // x-координата точки
    int nYPos,       // y-координата точки
    COLORREF clrref); // колір точки

```

Кожні 2 піксели змінюється колір виведених точок фігури. За їх зміну відповідає фрагмент коду:

```

.IF count==2      ; зміна кольору кожні X піксела
    @<inc colour>,<mov count,0>

```

```
.ELSE
  inc count
.ENDIF
```

Початковим кольором фігури вибрано зелений колір зі значенням:  
 colour dd 32000 ; зелений – 7D00,  
 відносно якого здійснюються такі зміни.

## 2.10. Створення двох вікон на прикладі фігури у вигляді рози

Полярна роза ([http://ru.wikipedia.org/wiki/Полярная\\_система\\_координат](http://ru.wikipedia.org/wiki/Полярная_система_координат)) – математична крива, схожа на квітку з пелюстками. Вона може бути визначена простим рівнянням у полярних координатах:

$$r(\varphi) = a \cos(k\varphi + \theta_0)$$

для довільної постійної  $\theta_0$  (включаючи 0). Якщо  $k$  – ціле число, то це рівняння визначатиме розу з  $k$  пелюстками для непарних  $k$  або з  $2k$  пелюстками для парних  $k$ . Якщо  $k$  – раціональне, але не ціле, то графік, який заданий рівнянням, утворює фігуру, подібну до троянди, але пелюстки перекриватимуться. Рози з 2, 6, 10, 14 і так далі пелюстками цим рівнянням визначити неможливо.

Якщо вважати, що радіус не може бути негативним, то при будь-якому натуральному  $k$  виходить пелюсткова роза. Таким чином, рівняння  $r(\varphi) = a \cos 2\varphi$  визначатиме розу з двома пелюстками. З геометричної точки зору радіус – це відстань від полюса до точки, і він не може бути негативним.

У програмі створюються **два вікна**: у першому вікні виконується рисування фігури рози, а в другому вікні виводиться фотографія.

**Програма 2.6.** Файл ресурсів програми створення рози:

```
#define IDM_EXIT 1 // "Выход"
#define IDM_AUTHOR 2 // "Об авторе"
#define IDM_HELP 3 // "О программе"
#define IDM_12 4 // Масштаб 1:2
#define IDM_11 5 // Масштаб 1:1
#define IDM_21 6 // Масштаб 2:1
#define IDM_COL1 7 // "Синий"
#define IDM_COL2 8 // "Зеленый"
#define IDM_COL3 9 // "Красный"
#define IDM_COL4 10 // "Черный"
#define IDM_FOTO 11 //
#define IDM_K1 13 // Кількість пелюстків – 1
#define IDM_K2 14 // Кількість пелюстків – 2
#define IDM_K3 15 // Кількість пелюстків – 3
#define IDM_K4 16 // Кількість пелюстків – 4
```

```

#define IDM_K5 17 // Кількість пелюстків – 5
#define IDM_K6 18 // Кількість пелюстків – 6
#define IDM_K7 19 // Кількість пелюстків – 7
#define IDL_ICON 22 //
#define IDB_PIC 23 //
IDB_PIC BITMAP "foto.bmp"

```

IDI\_ICON ICON DISCARDABLE MOVEABLE LOADONCALL "Roza.ico"

```

FirstMenu MENU {
    POPUP "Программа"{
        MENUITEM "О программе",IDM_HELP
        MENUITEM SEPARATOR
        MENUITEM "Выход",IDM_EXIT
    }
    POPUP "Параметры"{
    POPUP "Масштаб"{
        MENUITEM "1:2", IDM_12
        MENUITEM "1:1", IDM_11
        MENUITEM "2:1", IDM_21
    }
    POPUP "Цвет"{
        MENUITEM "Красный", IDM_COL1
        MENUITEM "Зеленый", IDM_COL2
        MENUITEM "Синий", IDM_COL3
        MENUITEM "Черный", IDM_COL4
    }
    POPUP "Кол-во лепестков"{
        MENUITEM "1", IDM_K1
        MENUITEM "2", IDM_K2
        MENUITEM "3", IDM_K3
        MENUITEM "4", IDM_K4
        MENUITEM "5", IDM_K5
        MENUITEM "6", IDM_K6
        MENUITEM "7", IDM_K7
    }
}
    POPUP "Автор"{
        MENUITEM "Об авторе", IDM_AUTOR
        MENUITEM "Фото", IDM_FOTO
    }
}

```

### Програма 2.7. Створення рози:

.686 ; директива визначення типу мікропроцесора  
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows

option casemap:none ; відмінність малих та великих літер  
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD ; прототип процедури

```
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
uselib masm32, user32, kernel32,gdi32, fpu,shell32
.data ; директива визначення даних
    ClassName db "Class", 0
    ClassName2 db "Class2", 0
    AppName db "Роза",0
    ChildName db "Фото автора",0
    MenuName db "FirstMenu",0
    info_string db " КИТ ",0
    usl_string db "Курсовой проект по системному программированию.",0dh,0ah,
        "Тема: Построение по геометрической формуле розы
c",0dh,0ah,
        "изменением цвета, в трех вариантах масштаба.",0
    info_caption db "Автор программы",0
    usl_caption db "О программе",0
red RGBQUAD <255,0,0>
green RGBQUAD <0,255,0>
blue RGBQUAD <0,0,255>
black RGBQUAD <0,0,0>
col dd 0
arg dt 0.01 ; аргумент збільшення кута
pi dt ? ; константа
fi dt 0 ; вугол фі
ci dt 200.0
k dt 5.0
k1 dt 1.0
k2 dt 1.5
k3 dt 2.0
k4 dt 2.5
k5 dt 3.0
k6 dt 3.5
k7 dt 4.0
tw dt 2.0 ; два
temp dt ? ; тимчасова змінна
x dt ? ; x для розрахунків
y dt ? ; y для розрахунків
xr dd 0 ; x для рисування
yr dd 0 ; y для рисування
cicl dd ? ; змінна збереження потокової кількості циклів two dd 2 ; два
xt dd 0 ; x для переведення
```

yt dd 0 ; у для переведення  
     xdiv2 dd ? ; середина за X  
     ydiv2 dd ? ; середина за Y  
     ci1 dt 100.0 ; масштабний коефіцієнт  
     ci2 dt 200.0 ; масштабний коефіцієнт  
     ci3 dt 300.0 ; масштабний коефіцієнт  
 hchild dd ? ; дескриптор дочірнього вікна  
 hBitmap dd ? ; дескриптор зображення  
 hwnd dd ? ; дескриптор батьківського вікна  
**note** NOTIFYICONDATA <> ; структура іконки на панелі tray  
**color** RGBQUAD <>

.data? ; директива невизначених даних  
     hInstance HINSTANCE ?  
     CommandLine LPSTR ?

.const ; визначення констант  
     IDM\_EXIT equ 1 ; "Выход"  
     IDM\_AUTHOR equ 2 ; "Об авторе"  
     IDM\_HELP equ 3 ; "О программе"  
     IDM\_12 equ 4 ; Масштаб 1:2  
     IDM\_11 equ 5 ; Масштаб 1:1  
     IDM\_21 equ 6 ; Масштаб 2:1  
     IDM\_COL1 equ 7 ; "Синий"  
     IDM\_COL2 equ 8 ; "Зеленый"  
     IDM\_COL3 equ 9 ; "Красный"  
     IDM\_COL4 equ 10 ; "Черный"  
 IDM\_FOTO equ 11 ;  
 IDM\_K1 equ 13 ; Кількість пелюстків – 1  
 IDM\_K2 equ 14 ; Кількість пелюстків – 2  
 IDM\_K3 equ 15 ; Кількість пелюстків – 3  
 IDM\_K4 equ 16 ; Кількість пелюстків – 4  
 IDM\_K5 equ 17 ; Кількість пелюстків – 5  
 IDM\_K6 equ 18 ; Кількість пелюстків – 6  
 IDM\_K7 equ 19 ; Кількість пелюстків – 7  
     IDI\_ICON equ 22  
     IDB\_PIC equ 23  
     IDI\_TRAY equ 0  
     WM\_SHELLNOTIFY equ 5

.code ; директива початку сегмента команд  
 start: ; мітка початку програми з ім'ям start

invoke GetModuleHandle, NULL ; отримання дескриптора програми  
     mov hInstance, eax ; збереження дескриптора програми



```
        invoke GetCommandLine
        mov CommandLine, eax
invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT
invoke ExitProcess, eax ; повернення управління ОС та визволення
ресурсів
```

### **WinMain proc**

```
hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
```

```
LOCAL wc:WNDCLASSEX ; резервування стека під структуру
LOCAL wc2:WNDCLASSEX ; резервування стека під структуру
LOCAL msg:MSG ; резервування стека під структуру MSG
```

```
mov wc.cbSize, SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
mov wc.lpszWndProc, OFFSET WndProc ; адреса процедури WndProc
mov wc.cbClsExtra, NULL ; кількість байтів для структури класу
mov wc.cbWndExtra, NULL ; кількість байтів для структури вікна
push hInst ; збереження в стеку дескриптора програми
pop wc.hInstance ; повернення дескриптора в поле структури
mov wc.hbrBackground, COLOR_WINDOW+1 ; колір вікна
mov wc.lpszMenuName, OFFSET MenuName ; ім'я ресурсу меню
mov wc.lpszClassName, OFFSET ClassName ; ім'я класу
```

```
invoke LoadIcon, hInstance, IDI_ICON ; ресурс піктограми
mov wc.hIcon, eax ; дескриптор піктограми
mov wc.hIconSm, eax ; дескриптор маленького віконця
```

```
invoke LoadCursor, NULL, IDC_ARROW ; ресурс
mov wc.hCursor, eax ; дескриптор курсора
```

```
invoke RegisterClassEx, addr wc ; реєстрація класу вікна
```

```
INVOKE CreateWindowEx, 0, ADDR ClassName, \ ; стиль, адр. імені класу
ADDR AppName, \ ; адреса імені вікна
WS_OVERLAPPEDWINDOW, \ ; базовий стиль
200, 200, 600, 600, \ ; координати вікна
0, 0, hInst, 0 ; дескриптори батьківського вікна, меню, програми
```

```
mov hwnd, eax
```

### **; для завантаження фото**

```
mov wc2.cbSize, SIZEOF WNDCLASSEX ; кількість байтів структури
mov wc2.style, CS_BYTEALIGNWINDOW ; стиль та поведінка вікна
```

```

mov wc2.lpfWndProc,OFFSET WndProc2 ; адреса процедури WndProc
mov wc2.cbClsExtra,NULL ; кількість байтів для структури класу
mov wc2.cbWndExtra,NULL ; кількість байтів для структури вікна
push hInst ; збереження в стеку дескриптора програми
pop wc2.hInstance ; повернення дескриптора в поле структури
mov wc2.hbrBackground,COLOR_BTNFACE+1; колір вікна
    mov wc2.lpszMenuName,NULL ; ім'я ресурсу меню
    mov wc2.lpszClassName,OFFSET ClassName2 ; ім'я класу

```

```

invoke LoadIcon,hInstance,IDI_ICON ; ресурс піктограми
    mov wc2.hIcon,eax ; дескриптор піктограми
    mov wc2.hIconSm,eax ; дескриптор маленького віконця

```

```

invoke LoadCursor,NULL, IDC_ARROW ; ресурс
    mov wc2.hCursor,eax ; дескриптор курсора

```

```

invoke RegisterClassEx,addr wc2 ; реєстрація класу вікна
INVOKE ShowWindow, hwnd,SW_SHOWNORMAL ; видимість вікна
INVOKE UpdateWindow, hwnd

```

```

    .WHILE TRUE ; поки істинне, то
    invoke GetMessage, ADDR msg, NULL, 0, 0 ; читання повідомлення
        or eax, eax ; формування ознак
        jz Quit ; перейти на мітку Quit, якщо eax = 0
    invoke DispatchMessage, ADDR msg ; відправлення на обслуговування
    .ENDW ; закінчення циклу оброблення повідомлень

```

Quit:

```

    mov eax,msg.wParam
    ret ; повернення з процедури WinMain
WinMain endp ; закінчення процедури з ім'ям WinMain

```

```

WndProc proc hWnd:HWND, uMsg:UINT,
wParam:WPARAM, lParam:LPARAM
LOCAL rect:RECT ; резервування стека під структуру RECT
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
LOCAL hdc:HDC ; резервування стека під хендл вікна

```

```

.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
invoke PostQuitMessage,NULL ; передача повідомлення WM_Quit

```

```

.ELSEIF uMsg==WM_CREATE
    mov color.rgbRed,0
    mov color.rgbGreen,0
    mov color.rgbBlue,0

```

```

mov note.cbSize, sizeof NOTIFYICONDATA ; розмір структури
push hwnd
pop note.hwnd ; хендл вікна
mov note.uID, IDI_TRAY ; ідентифікатор іконки
mov note.uFlags, NIF_ICON+NIF_MESSAGE+NIF_TIP ; всі поля
структури
mov note.uCallbackMessage, WM_SHELLNOTIFY ; користувацьке
повідомлення
invoke LoadIcon, hInstance, IDI_ICON ; завантаження хендла іконки
mov note.hIcon, eax
invoke IStrcpy, ADDR note.szTip, ADDR AppName ; висхідне підказування
invoke Shell_NotifyIcon, NIM_ADD, ADDR note ; додавання іконки в трей

.ELSEIF uMsg==WM_COMMAND ; якщо є повідомлення з меню
mov eax, wParam ; формування ознак повідомлення з меню

.IF ax==IDM_AUTOR ; виведення інформації про автора
invoke MessageBox, NULL, ADDR info_string, ADDR info_caption, MB_OK

.ELSEIF ax==IDM_HELP ; виведення умови
invoke MessageBox, NULL, ADDR usl_string, ADDR usl_caption, MB_OK

.ELSEIF ax==IDM_12 ; Масштаб 1:2
invoke FpuSub, ADDR ci, ADDR ci, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke FpuAdd, ADDR ci, ADDR ci1, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

.ELSEIF ax==IDM_11 ; Масштаб 1:1
invoke FpuSub, ADDR ci, ADDR ci, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke FpuAdd, ADDR ci, ADDR ci2, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

.ELSEIF ax==IDM_21 ; Масштаб 2:1
invoke FpuSub, ADDR ci, ADDR ci, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke FpuAdd, ADDR ci, ADDR ci3, ADDR ci, SRC1_REAL or SRC2_REAL
or DEST_MEM
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

.ELSEIF ax==IDM_K1 ; кількість пелюстків – 1
invoke FpuSub, ADDR k, ADDR k, ADDR k, SRC1_REAL or SRC2_REAL or
DEST_MEM

```

invoke FpuAdd, ADDR k,ADDR k1,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K2** ; кількість пелюстків – 2

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k2,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K3** ; кількість пелюстків – 3

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k3,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K4** ; кількість пелюстків – 4

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k4,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K5** ; кількість пелюстків – 5

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k5,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K6** ; кількість пелюстків – 6

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k6,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_K7** ; кількість пелюстків – 7

invoke FpuSub, ADDR k,ADDR k,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke FpuAdd, ADDR k,ADDR k7,ADDR k, SRC1\_REAL or SRC2\_REAL or DEST\_MEM

invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_COL1** ; установлення кольору та WM\_PAINT  
mov color.rgbRed,0  
mov color.rgbGreen,0  
mov color.rgbBlue,255  
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_COL2** ; встановлення кольору та WM\_PAINT  
mov color.rgbRed,0  
mov color.rgbGreen,255  
mov color.rgbBlue,0  
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_COL3** ; установлення кольору та WM\_PAINT  
mov color.rgbRed,255  
mov color.rgbGreen,0  
mov color.rgbBlue,0  
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_COL4** ; встановлення кольору WM\_PAINT  
mov color.rgbRed,0  
mov color.rgbGreen,0  
mov color.rgbBlue,0  
invoke InvalidateRect, hWnd, NULL, TRUE ; перерисовування вікна

**.ELSEIF ax==IDM\_FOTO**  
INVOKE CreateWindowEx,\ ; функція створення вікна за зразком  
NULL,ADDR ClassName2,\ ; стиль та адреса імені класу  
ADDR ChildName,\ ; адреса імені вікна  
WS\_OVERLAPPEDWINDOW,\ ; базовий стиль  
200,200,350,400,\ ; координати вікна

hwnd,0,hInstance,0 ; дескриптори батьківського вікна, меню, програми  
mov hchild,eax  
INVOKE ShowWindow, hchild,SW\_SHOWNORMAL ; видимість вікна  
INVOKE UpdateWindow, hchild

**.ELSE**  
invoke DestroyWindow,hWnd ; знищення вікна

**.ENDIF**

**.ELSEIF uMsg==WM\_PAINT** ; якщо є повідомлення про  
перерисовування  
invoke BeginPaint, hWnd, addr ps ; підготовка проц. та заповнення структ.  
mov hdc, eax ; збереження контексту  
invoke GetClientRect, hWnd, addr rect; занесення характ. вікна

invoke FpuSub, ADDR fi, ADDR fi, ADDR fi, SRC1\_REAL or SRC2\_REAL or DEST\_MEM ; встановлення кута на 0

```
finit          ; ініціювання співпроцесора
fld rect.right ; st(0) = (ширина вікна)
fld two       ; st(0) = 2
fdivp st(1), st ; st(0) = (ширина вікна)/2
fistp dword ptr xdiv2 ; збереження середини за X
fld rect.bottom ; st(0) = (висота вікна)
fld two       ; st(0) = 2
fdivp st(1), st ; st(0) = (висота вікна)/2
fistp dword ptr ydiv2 ; збереження середини за Y
mov ecx, 36000 ; встановлення кількості циклів
mov cicl, ecx  ; занесення кількості циклів у змінну
```

I1:

invoke FpuMul, ADDR k, ADDR fi, 0, SRC1\_REAL or SRC2\_REAL or DEST\_FPU

invoke FpuSin, 0,0, SRC1\_FPU or DEST\_FPU

invoke FpuMul, 0, ADDR ci, ADDR pi, SRC1\_FPU or SRC2\_REAL or DEST\_MEM

invoke FpuCos, ADDR fi,0, SRC1\_REAL or DEST\_FPU ;st(0)=cos(fi)

invoke FpuMul, 0, ADDR pi, ADDR x, SRC1\_FPU or SRC2\_REAL or DEST\_MEM

;  $x = \pi * \cos(fi)$

invoke FpuSin, ADDR fi,0, SRC1\_REAL or DEST\_FPU ; st(0)=sin(fi)

invoke FpuMul, 0, ADDR pi, ADDR y, SRC1\_FPU or SRC2\_REAL or DEST\_MEM

;  $x = \pi * \sin(fi)$

fld x

frndint ; округлення st(0)

fistp dword ptr xt

fld y

frndint ; округлення st(0)

fistp dword ptr yt

; зміщення точки відповідно до центру вікна

mov eax,xt

add eax,xdiv2

mov xr,eax

mov eax,yt

add eax,ydiv2

mov yr,eax

invoke SetPixel, hdc, xr, yr, dword ptr color ; рисування точки

invoke FpuAdd, ADDR fi, ADDR arg, ADDR fi, SRC1\_REAL or SRC2\_REAL or DEST\_MEM ;кут

```

mov ecx,cicl ; занесення кількості циклів у регістр
dec ecx ; зменшення лічильника
mov cicl,ecx ; занесення кількості циклів у змінну
push ecx ; занесення кількості циклів у стек
jz I2 ; вихід з циклу
jmp I1 ; продовження рисування

```

I2:

```

invoke EndPaint, hWnd, addr ps ; закінчення рисування
mov eax, 0

```

```

.ELSE ; інакше
invoke DefWindowProc,hWnd,uMsg,wParam,IParam
ret
.ENDIF ; закінчення логічної структури .IF - .ELSE
xor eax, eax
ret ; повернення з процедури

```

**WndProc endp** ; закінчення процедури WndProc

**WndProc2 proc** hWin:DWORD,uMsg:DWORD,wParam :DWORD,IParam :DWORD

```

LOCAL rect:RECT ; резервування стека під структуру RECT
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
LOCAL hdc:HDC ; резервування стека під хендл вікна
LOCAL hMemDC:HDC ; резервування стека під хендл вікна

```

**.if uMsg == WM\_CREATE**

```

invoke LoadBitmap,hInstance,IDB_PIC ;отримання хендла зображення
mov hBitmap,eax

```

**.elseif uMsg == WM\_PAINT**

```

invoke BeginPaint,hchild,ADDR ps ; виклик процедури та заповнення ps
mov hdc,eax
invoke CreateCompatibleDC,hdc ; створення контексту пам'яті
mov hMemDC,eax
invoke SelectObject,hMemDC,hBitmap ; вибір у контекст
invoke GetClientRect,hchild,ADDR rect ; визначення розміру вікна
invoke BitBlt,hdc,0,0,rect.right,rect.bottom,hMemDC,0,0,SRCCOPY ;
виведення
invoke DeleteDC,hMemDC ; видалення дескриптора
invoke EndPaint,hchild,ADDR ps ; закінчення рисування

```

**.elseif uMsg == WM\_DESTROY**

```

invoke DeleteObject,hBitmap
invoke PostQuitMessage,NULL ; передача повідомлення WM_Quit

```

```

.endif
invoke DefWindowProc,hWin,uMsg,wParam,lParam
ret
WndProc2 endp
end start ; закінчення програми з ім'ям start

```

Результат виконання програми наведено на рис. 2.9.

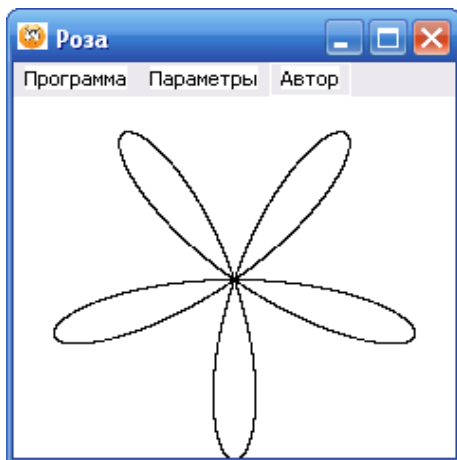


Рис. 2.9. Результат виконання програми

## 2.11. Лабораторна робота “Графічні примітиви”

### Мета заняття:

- поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;
- набути практичних навичок віконного програмування на асемблері для платформи x86.

### Постановка задачі

Написати програму зі створенням *декількох вікон* з відображенням маленької іконки вікна, не менше ніж з чотирьох пунктів головного меню, одним з яких є очищення вікна, використати фон вікна, фон заливки геометричних фігур, кольоровий шрифт з назвою свого прізвища, навчальної групи та завдання, рамки навколо фігури, файлу ресурсів меню й завданням згідно з варіантом:

1. Відобразити вписані один в один трикутник та прямокутник.



2. Відобразити вписані один в один два закруглених прямокутника та трикутники.

3. Відобразити вписані один в один еліпс та закруглений прямокутник.

4. Відобразити 4 кола, які доторкуються кутів прямокутника.

5. Відобразити по колу різнокольорові кола з однаковим діаметром.

6. Відобразити 3 кола, які доторкуються кутів трикутника.

7. Відобразити п'ятикутник з вписаним колом.

8. Відобразити п'ятикутну зірку.

9. Відобразити по колу різнокольорові кола з різними діаметрами.

10. Відобразити вписані один в один чотирикутники.

11\*. Відобразити вікно, розміри якого плавно змінюються.

Розглянемо програму виведення закругленого прямокутника та трикутника. Файл ресурсів програми наведено в програмі 2.8.

**Програма 2.8.** Файл ресурсів програми:

```
#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_AUTOR 3
#define IDM_HELP 4
#define IDM_CLEAN 5
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL "Gr1.ico"
FirstMenu MENU {
    POPUP "Фигура"{
        MENUITEM "Фигуры",IDM_HELLO
        MENUITEM SEPARATOR
        MENUITEM "Очистить", IDM_CLEAN
        MENUITEM SEPARATOR
        MENUITEM "Выход",IDM_EXIT
    }
    MENUITEM "Автор",IDM_AUTOR
    POPUP "HELP"{
        MENUITEM "Help",IDM_HELP
    }
}
```

У кнопці **Автор** відсутня висхідна кнопка, а при натисканні кнопки виводиться текст у вікно.

По кнопці **Help** виводиться повідомлення у спрощене віконце через функцію **MessageBox**.

Результаты работы программы наведены на рис. 2.10.

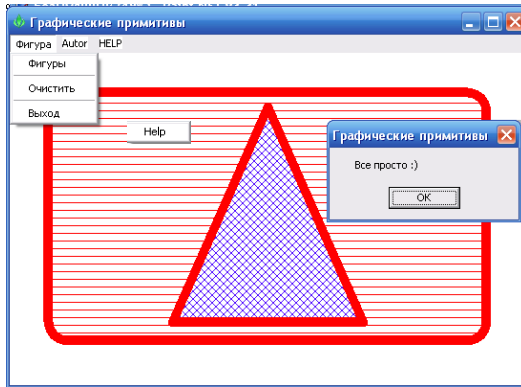


Рис. 2.10. Результаты работы программы

Программа вывода закругленного прямоугольника та трикутника наведена в програмі 2.9.

**Программа 2.9.** Вывод закругленного прямоугольника та трикутника:

```
title Rysovaniy A.N.
.686          ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\include\windows.inc
include \masm32\macros\macros.asm
uselib user32,kernel32,gdi32
```

**@ MACRO b0,b1,b2,b3**

```
b0
b1
b2
b3
ENDM
```

.data

```
poln POINT <270,55>, <170,280>, <370,280>
mytext db "Рысованный А.Н., e-mail: rysov@rambler.ru",0
ClassName db "SimpleWinClass",0
AppName db "Графические примитивы",0
```

```

MenuName db "FirstMenu",0
Help_string db " Все просто :)",0
color1 RGBQUAD <255,0,0>
color2 RGBQUAD <80,10,255>
color3 RGBQUAD <0,110,255>
hPen1 dd 0 ; дескриптор створюваного пера
hOldPen dd 0 ; дескриптор початкового пера

.data? ; директива невизначених даних
hInstance HINSTANCE ?
CommandLine LPSTR ?
blueBrush dd ? ; комірка для параметрів кисті

.const
IDM_FUNC equ 1
IDM_EXIT equ 2
IDM_AUTHOR equ 3
IDM_HELP equ 4
IDM_CLEAN equ 5
IDI_ICON equ 22

.code ; директива початку сегмента команд
start: ; мітка початку програми з ім'ям start
invoke GetModuleHandle, NULL ; отримання дескриптора програми
mov hInstance,eax ; збереження дескриптора програми
invoke GetCommandLine
mov CommandLine,eax
invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
invoke ExitProcess,eax

```

```

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
CmdShow:DWORD
LOCAL wc:WNDCLASSEX ; резервування стека під структуру
LOCAL msg:MSG ; резервування стека під структуру MSG
LOCAL hwnd:HWND ; резервування стека під хендл програми
@<mov wc.cbSize,SIZEOF WNDCLASSEX>,<mov wc.cbClsExtra,0>
@<mov wc.style,CS_HREDRAW or CS_VREDRAW >,<mov wc.cbWndExtra,0>
@<mov wc.lpfWndProc,OFFSET WndProc>,<push hInst>,<pop wc.hInstance>
@<mov wc.hbrBackground,COLOR_WINDOW+1>,<mov wc.lpszMenuName,OFFSET MenuName>
mov wc.lpszClassName,OFFSET ClassName ; ім'я класу
invoke LoadIcon,hInstance, IDI_ICON ; ресурс піктограми
@< mov wc.hIcon,eax >,< mov wc.hIconSm,eax >
invoke LoadCursor,NULL,IDC_ARROW ; ресурс курсора
mov wc.hCursor,eax

```

invoke RegisterClassEx, addr wc ; реєстрація класу вікна  
 invoke CreateWindowEx, \ ; функція створення вікна за зразком  
 0,ADDR ClassName, ADDR AppName, \ ; стиль та адреса імен класу та  
 вікна  
 WS\_OVERLAPPEDWINDOW, \ ; базовий стиль  
 CW\_USEDEFAULT,CW\_USEDEFAULT,\; гориз. та верт. координати ві-

кна  
 550,400, \ ; ширина та висота вікна  
 0,0,hInst,0 ; дескриптори батьківського вікна, меню, програми

mov hwnd,eax  
 INVOKE ShowWindow, hwnd,SW\_SHOWNORMAL  
 INVOKE UpdateWindow, hwnd

.WHILE TRUE  
 INVOKE GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення  
 .BREAK .IF (!eax)  
 INVOKE DispatchMessage, ADDR msg ; відправлення на обслуговування  
 .ENDW  
 mov eax,msg.wParam  
 ret ; повернення з процедури WinMain  
**WinMain endp** ; закінчення процедури з ім'ям WinMain

### **WndProc proc**

hWnd:HWND,uMsg:UINT,wParam:WPARAM,IParam:LPARAM  
 LOCAL hdc:HDC ; резервування стеку під хендл вікна

#### **.IF uMsg==WM\_DESTROY**

invoke PostQuitMessage,NULL

#### **.ELSEIF uMsg==WM\_CREATE** ; оброблення повідомлення

WM\_CREATE

invoke CreatePen,\ ; створення пера

PS\_SOLID,10,\ ; завтовшки 10 пікселів

dword ptr color1 ;

mov hPen1,eax; збереження отриманого дескриптора пера

#### **.ELSEIF uMsg==WM\_COMMAND** ; якщо є повідомлення від меню

mov eax,wParam

#### **.IF ax==IDM\_FUNC** ; якщо є повідомлення про відображення фігур

invoke GetDC, hWnd ; отримати дескриптор контексту пристрою

mov hdc, eax ; збереження

invoke SelectObject, hdc,hPen1 ; вибирається в контекст нове перо

invoke CreateHatchBrush, \ ; створення кисті для зарисовування фігури

HS\_HORIZONTAL,\ ; стиль – горизонтальна штриховка

```

    dword ptr color1 ; значення кольору
    mov blueBrush,eax ; збереження параметрів кисті
invoke SelectObject,\ ; вибір об'єкта з параметрами
    hdc, blueBrush ; контекст пристрою та параметри кисті
invoke RoundRect,hdc,\ ; дескриптор контексту пристрою
    40,40,\ ; x-, y-координата верх. лівого кута
    500,300,\ ; x-, y-коорд. нижн. прав кута
    40,40 ; ширина та висота еліпса для рисування кута
invoke CreateHatchBrush, \ ; створення кисті для зарисовування фігури
    HS_DIAGCROSS,\ ; стиль – штриховка під 45 зліва та справа
    dword ptr color2 ; значення кольору
    mov blueBrush,eax ; збереження параметрів кисті
invoke SelectObject,\ ; вибір об'єкта з параметрами
    hdc,\ ; контекст пристрою
    blueBrush ; параметри кисті
invoke Polygon, hdc, ;
    addr poln ; адреса масиву координат точок
    3 ; кількість вершин у масиві
    invoke ReleaseDC, hWnd, hdc ; визволення контексту пристрою

```

```

.ELSEIF ax==IDM_AUTOR ; якщо є повідомлення "Autor"
invoke GetDC, hWnd ; отримання дескриптора контексту пристрою
    mov hdc, eax ; збереження
invoke SelectObject, hdc,hPen1 ; отримання в контекст нового пера
invoke SetTextColor,hdc, dword ptr color3 ; установлення кольору тексту
invoke TextOut,hdc, 170,10,\ ; x-, y-координата стартової позиції
    addr mytext, 40 ; адреса рядка та число символів у рядку

```

```

.ELSEIF ax==IDM_CLEAN ; очистити
invoke InvalidateRect, hWnd, NULL, TRUE ; виклик функції та WM_PAINT

```

```

. ELSEIF ax==IDM_HELP ; якщо є повідомлення "Help"
    invoke MessageBox,NULL,ADDR Help_string,OFFSET
    AppName,MB_OK

```

```

. ELSE
    invoke DestroyWindow,hWnd ; знищення вікна
.ENDIF ; закінчення логічної структури
.else
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; обробка та
    ; відправлення повідомлення до функції WndProc
    ret ; повернення з процедури
.endif ; закінчення логічної структури
    xor eax,eax
    ret ; повернення з процедури
    WndProc endp ; закінчення процедури WndProc

```

end start ; закінчення програми з ім'ям start

Розглянемо програму, яка виконує **очищення списку документів та очищення кошика**. Робота програми здійснюється таким чином:

1. На екран виводиться вікно з меню системи і з текстом призначення програми.

2. Оператор обирає режим роботи.

3. Здійснюється аналіз обраного режиму роботи. Залежно від обраного режиму викликається відповідна процедура, що реалізує обраний режим:

– очищення системи;

– відображення інформації про автора;

– відображення інформації про утиліту;

– закриття вікна.

Якщо обирається режим закінчення роботи, то здійснюється вихід із програми.

**Алгоритм очищення системи** полягає в наступному:

– відкривається вікно;

– отримується від вікна повідомлення;

– якщо повідомлення про натиснення кнопки «Очистити систему» в самому вікні або в меню «Програма», то викликається діалогове вікно з підтвердженням наступних дій;

– якщо отримується підтвердження, то викликається процедура очищення списку нещодавно відкритих документів та процедура очищення;

– якщо процедури виконані успішно, то виводиться діалогове вікно з відповідним повідомленням;

– якщо процедури не виконані, то виводиться відповідне повідомлення в діалоговому вікні.

Програма очищення системи наведена у програмі 2.10.

**Програма 2.10.** Очищення системи:

```
.686
```

```
.model flat,stdcall
```

```
option casemap:none
```

```
include \masm32\include\windows.inc ; файли структур, констант ...
```

```
include \masm32\include\windows.inc
```

```
include \masm32\macros\macros.asm
```

```
uselib user32, kernel32, gdi32, shell32
```

```
szText MACRO Name, Text:VARARG
```

```
LOCAL lbl
```

```

    jmp lbl
    Name db Text,0

    lbl:
ENDM

; оголошення констант елементів управління і ікони
    IDM_HELLO equ 1
    IDM_EXIT equ 2
    IDM_ABOUT equ 3
    IDM_HELP equ 4
    ID_BUTTON equ 9
    ID_ICON equ 22

.data
    wc WNDCLASSEX <>
    msg MSG <> ; структура повідомлень
    hwnd HWND ?
    hButton HWND ?
    hInstance HINSTANCE ?

.code ; директива початку сегмента команд
start: ; мітка початку програми з ім'ям start

    szText ButtonClassName,"button" ; оголошення змінної через макрос
    szText ButtonText, "Очистить систему"
    szText ClassName, "SimpleWinClass"
    szText MenuName, "FirstMenu"
    szText szMessage,"Вы действительно хотите очистить корзину и список
недавно открываемых документов?"
    szText szTitle, "Очистка корзины и списка документов: System Cleaner"
    szText szError, "Корзина и список недавно открываемых документов уже
очищены!",0
    szText szAbout, "Задание:",13,10,\
"Написать оконную программу под Win32, которая выполняет следующие
функции:",13,10,\
" - очистка корзины",13,10,\
" - очистка списка недавно открываемых документов",0
    szText szHelp,"Системное программирование",13,10,\
"студент группы КИТ-186, НТУ ХПИ",13,10,"Мричко Андрей",0
    szText szDone,"Очистка выполнена успешно!",0

    invoke GetModuleHandle, NULL ; отримання дескриптора програми
    mov hInstance,eax ; збереження дескриптора програми
    mov wc.cbSize,SIZEOF WNDCLASSEX ; кількість байтів структури
    mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
    mov wc.lpfWndProc, OFFSET WndProc ; адреса процедури WndProc

```

```

mov  wc.cbClsExtra,NULL      ; кількість байтів для структури класу
mov  wc.cbWndExtra,NULL     ; кількість байтів для структури вікна
push hInstance              ; збереження в стеку дескриптора програми
pop   wc.hInstance          ; повернення дескриптора в поле структури
mov  wc.hbrBackground,COLOR_WINDOW+1 ; колір вікна
mov  wc.lpszMenuName,OFFSET MenuName ; ім'я ресурсу меню
mov  wc.lpszClassName,OFFSET ClassName ; ім'я класу

```

```

invoke LoadIcon,hInstance,IDI_ICON ; відображення піктограми
mov  wc.hIcon,eax збереження дескриптора ікони
mov  wc.hIconSm,eax ; збереження дескриптора малої ікони

```

```

invoke LoadCursor,NULL,IDC_ARROW ; завантаження курсора
mov  wc.hCursor,eax

```

```

invoke RegisterClassEx, addr wc ; функція реєстрації класу вікна

```

```

invoke CreateWindowEx,NULL, ADDR ClassName,\
ADDR szTitle, WS_OVERLAPPEDWINDOW,\
450, 450, 666,175,NULL,NULL,hInstance,NULL
mov  hwnd,eax
invoke ShowWindow, hwnd,SW_SHOWNORMAL ; видимість вікна

```

```

.WHILE TRUE ; поки істинне, то
invoke GetMessage, ADDR msg,NULL,0,0 ; читання повідомлення
or  eax, eax ; формування ознак
jz  Quit ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправка до WndProc
.ENDW ; закінчення циклу оброблення повідомлень
Quit:
mov  eax,msg.wParam
invoke ExitProcess, eax

```

```

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,\
IParam:LPARAM

```

```

LOCAL hdc:HDC ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру
PAINTSTRUCT
LOCAL rect:RECT резервування стека під структуру RECT

```

```

.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
invoke PostQuitMessage,NULL ; передача повідомлення WM_Quit

```



**.ELSEIF uMsg==WM\_COMMAND** ; якщо є повідомлення від меню  
mov eax,wParam

**.IF ax==IDM\_HELLO** ; якщо є повідомлення IDM\_HELLO  
invoke MessageBox,0,ADDR szMessage,ADDR  
szTitle,MB\_ICONQUESTION or MB\_YESNO

**.IF eax==IDYES** ; якщо натиснута кнопка Так  
invoke SHAddToRecentDocs,SHARD\_PATH,NULL ; очищення списку  
; документів, що недавно відкривались  
invoke SHEmptyRecycleBin,0,0,  
SHERB\_NOCONFIRMATION+SHERB\_NOPROGRESSUI ; очищення корзини

**.IF eax == S\_OK** ; якщо процедури виконані успішно  
invoke MessageBox,NULL,ADDR szDone,ADDR szTitle,  
MB\_ICONINFORMATION+MB\_OK  
**.else**  
invoke MessageBox,NULL,ADDR szError,ADDR  
szTitle,MB\_ICONERROR+MB\_OK  
**.endif**  
**.ENDIF**

**.ELSEIF ax==IDM\_HELP**  
invoke MessageBox,NULL,ADDR szAbout,ADDR szTitle,  
MB\_ICONINFORMATION+MB\_OK

**.ELSEIF ax==IDM\_ABOUT**  
invoke MessageBox,NULL,ADDR szHelp, ADDR szTitle, \  
MB\_ICONINFORMATION+MB\_OK

**.ELSEIF ax==ID\_BUTTON**  
shr eax, 16

**.IF ax==BN\_CLICKED**  
invoke MessageBox,0,ADDR szMessage,ADDR  
szTitle,MB\_ICONQUESTION or MB\_YESNO  
**.IF eax==IDYES** ; якщо натиснута кнопка Так  
invoke SHAddToRecentDocs,SHARD\_PATH,NULL ; очищення списку  
; документів, що недавно відкривались  
invoke SHEmptyRecycleBin,0,0,  
SHERB\_NOCONFIRMATION+SHERB\_NOPROGRESSUI ; очищення корзини  
**.IF eax == S\_OK** ; якщо процедури виконані успішно  
invoke MessageBox,NULL,ADDR szDone,ADDR szTitle,  
MB\_ICONINFORMATION+MB\_OK  
**.else**  
invoke MessageBox,NULL,ADDR szError,ADDR szTitle,  
MB\_ICONERROR+MB\_OK

```

        .endif
    .ENDIF
.ENDIF
.ELSE
    invoke DestroyWindow,hWnd
.ENDIF

```

```

.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
invoke CreateWindowEx, NULL, ADDR ButtonszClassName,\
    ADDR ButtonText, WS_CHILD or WS_VISIBLE or BS_PUSHBUTTON,\
    250, 70, 140, 25, hWnd, ID_BUTTON, hInstance, NULL ; створення кнопки у вікні
    mov hButton, eax
invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури та
                                ; заповнення структури
    mov hdc,eax ; збереження контексту
invoke GetClientRect,hWnd,ADDR rect ; занесення до структури rect харак-
теристик вікна
invoke DrawText,hdc,ADDR szAbout,-1,ADDR rect, DT_TOP or DT_LEFT
    invoke EndPaint,hWnd,ADDR ps

```

```

.ELSE ; інакше
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; обробка та
                                ; відправлення повідомлення до функції WndProc
    ret ; повернення з процедури

```

```

.ENDIF ; закінчення логічної структури .IF – .ELSEIF
xor eax,eax
ret ; повернення з процедури
WndProc endp ; закінчення процедури WndProc
end start ; закінчення програми з ім'ям start

```

У програмі 2.11 наведено файл ресурсів програми очищення системи.

**Програма 2.11.** Файл ресурсів програми очищення системи:

```

#define IDM_HELLO 1
#define IDM_EXIT 2
#define IDM_ABOUT 3
#define IDM_HELP 4
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL
"sysclean_icon.ico"
    FirstMenu MENU {
        POPUP "Программа"{
            MENUITEM "Очистить систему",IDM_HELLO
            MENUITEM SEPARATOR

```

```

    MENUITEM "Выйти",IDM_EXIT
}
POPUP "Информация"{
    MENUITEM "Задание",IDM_HELP
    MENUITEM "Автор",IDM_ABOUT
}
}

```

На рис. 2.11 наведено вікна програми очищення системи.

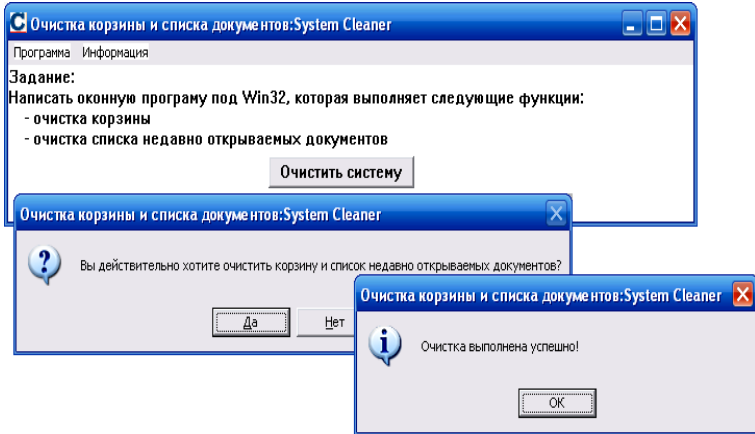


Рис. 2.11. Вікна програми очищення системи

## 2.12. Лабораторна робота “Графічні фігури”

### Мета заняття

набути практичних навичок віконного програмування на асемблері для платформи x86.

### Постановка задачі

Написати програму зі створенням *декількох вікон* з відображенням маленької іконки вікна, не менше ніж з чотирьох пунктів головного меню, одним з яких є очищення вікна, використати фон вікна, фон заливки геометричних фігур, кольоровий шрифт з назвою свого прізвища,

навчальної групи та завдання, рамки навколо фігури, файлу ресурсів меню й завданням згідно з варіантом:

1. Нарисувати зображення “Мережива”. Для цього на екрані будуть вершини правильного 18-кутника, центр якого збігається з центром екрана. Кожна з вершин з’єднується відрізками зі всіма іншими вершинами. Координати вершин задаються за формулами:

$$X_i = X_c + R \cos(2\pi i/n);$$
$$Y_i = Y_c + R \sin(2\pi i/n); \quad i = 1 \div 18,$$

де  $i$  – номер вершини;  $R$  – радіус кола, описаного біля багатокутника;  $X_c$ ,  $Y_c$  – координати центра.

2. Нарисувати повернутий еліпс.

(<http://rusproject.narod.ru/article/polar.htm/>)

Рівняння еліпса (<http://rusproject.narod.ru/article/polar.htm/>):

$$X = a \cdot \cos A$$

$$Y = b \cdot \sin A$$

Рівняння повернутого еліпса:

$$x1 = x \cdot \cos(F) + y \cdot \sin(F)$$

$$y1 = -x \cdot \sin(F) + y \cdot \cos(F),$$

де  $A$  – кут (0-360);  $a$ ,  $b$  – напівосі еліпса;  $F$  – кут повороту еліпса;  $X$ ,  $Y$  – координати на еліпсі до повороту;  $x1$ ,  $y1$  – координати на еліпсі після повороту (<http://forum.sources.ru/index.php?showtopic=117690>).

3. Нарисувати циклоїдальну криву у вигляді завитка Паскаля за параметричними рівняннями:

$$x = (R + mR) \cos(mt) - h \cos(t + mt);$$

$$y = (R + mR) \sin(mt) - h \sin(t + mt), \quad \text{при } R = 1; r = 1; h = 1.5,$$

де  $R$  – радіус нерухомого кола;  $r$  – радіус кола, що котиться;  $h$  – відстань від центра кола, що котиться, до точки;  $m = r/R$ .

4. Нарисувати циклоїдальну криву у вигляді рози за параметричними рівняннями:

$$x = (R + mR) \cos(mt) - h \cos(t + mt);$$

$$y = (R + mR) \sin(mt) - h \sin(t + mt), \quad \text{при } R = 1; r = 1/6; h = 1 + 1/6,$$

де  $R$  – радіус нерухомого кола;  $r$  – радіус кола, що котиться;  $h$  – відстань від центра кола, що котиться, до точки;  $m = r/R$ .

5. Нарисувати циклоїдальну криву у вигляді епіциклоїди за параметричними рівняннями:

$$x = (R + mR) \cos(mt) - m \cos(t + mt);$$

$$y = (R + mR) \sin(mt) - m \sin(t + mt), \quad \text{при } m = 1/10, 1/3, 2/3,$$

де  $R$  – радіус нерухомого кола;  $r$  – радіус кола, що котиться;  $m = r/R$ .

6. Нарисувати циклоїдальну криву у вигляді гіпоциклоїди за параметричними рівняннями:

$$x = (R - mR)\cos(mt) + m\cos(t - mt);$$

$$y = (R - mR)\sin(mt) - m\sin(t - mt), \text{ при } m = 1/4, 1/2, 1,$$

де  $R$  – радіус нерухомого кола;  $r$  – радіус окружності, що котиться;  $m = r/R$ .

7. Нарисувати циклоїдальну криву у вигляді гіпоциклоїди за параметричними рівняннями:

$$x = R\cos^3\Theta;$$

$$y = R\sin^3\Theta,$$

де  $R$  – радіус кола.

8. Нарисувати згідно з формулами кардіоїду. Передбачити статичний та динамічний режими, масштаб відображення, фон екрана та колір фігури (фігур з різними параметрами). Координати вершин у параметричних координатах задаються за формулами:

$$X = 2R\cos(1 + \cos t);$$

$$Y = 2R\sin t(1 + \cos t),$$

де  $R$  – радіус кола.

9. Нарисувати фігуру Ліссажу за формулами:

$$X(t) = A\sin(at + \delta);$$

$$Y(t) = B\sin(bt),$$

де  $A, B$  — амплітуди коливань;  $a, b$  — частоти;  $\delta$  — зсув фаз;  $a/b$  змінюється від 0 до 13 кроком 0.01 ( $\delta = 0$ ).

10. Нарисувати фігуру Ліссажу за формулами:

$$X(t) = A\sin(at + \delta);$$

$$Y(t) = B\sin(bt),$$

де  $A, B$  — амплітуди коливань,  $a, b$  — частоти,  $\delta$  — зсув фаз,

Число  $a$  – непарне натуральне, а також натуральне число  $b$ ;  $|a - b| = 1$ . ( $\delta = \pi/2$ ) (<https://ru.wikipedia.org/> Фигуры Лиссажу).

11. Нарисувати декілька зображень Лемніскату Бернуллі за формулами:

$$X(t) = c(\sqrt{2})(p + p^3)/(1 + p^4);$$

$Y(t) = c(\sqrt{2})(p - p^3)/(1 + p^4)$ , де  $p = \tan(\pi/4 - \phi)$ ,  $c$  – відстань між фокусами. ([https://ru.wikipedia.org/wiki/Лемниската\\_Бернулли](https://ru.wikipedia.org/wiki/Лемниската_Бернулли))

12\*. Дослідити овали Кассіні  
(<http://rusproject.narod.ru/arbuz/cassini.htm>).

### **Контрольні запитання**

1. Яке повідомлення відповідає за перерисовування вікна?
2. У чому полягає значення контексту пристрою та які він має типи?
3. У якому контексті не потрібно повторно здійснювати дії щодо зміни контексту?
4. Яку товщину в пікселях має стандартне перо та яке воно має ім'я?
5. Яка функція відповідає за зміну кольору пера?

### 3. ІНТЕРФЕЙС GDI+

GDI+ (Graphic Device Interface+ – інтерфейс графічних пристроїв) – це підсистема Microsoft Windows XP та .NET Server, що забезпечує виведення графічної інформації на екрани та принтери. GDI+ є наступником GDI, інтерфейсу графічних пристроїв, що включена в ранні версії Windows.

Можливості GDI+-технології (<http://www.rsdn.ru/article/gdi/gdiplus1.xml>) такі:

- градієнтне зафарбовування;
- підтримка прозорості;
- використання сплайнів: окрім кривих Безьє, підтримується новий вигляд кривих – так звані сплайни, які імітують поведінку натягнутої і зігнутої сталевий смуги. Сплайни є гладкими кривими;
- використання шляхів: шляхи існують незалежно від контексту рисування і є засобом створення складних векторних об'єктів;
- використання координатних перетворень: об'єкт Matrix дозволяє здійснювати операції повороту, перенесення, масштабування і віддзеркалення об'єктів GDI+;
- використання регіонів: на відміну від GDI, регіони не прив'язані до координат пристрою і підпорядковуються координатним перетворенням;
- використання растрів: підтримуються растри з накладенням зовнішнього альфа-каналу (для прозорості), масштабуванням, розтягуванням, спотворенням і поворотом растрів. При цьому можна встановити режими відображення окремих пікселів;
- підтримка популярних форматів графічних файлів.

#### 3.1. Інтерфейс керованих класів

Доступ до API-інтерфейсу в GDI+ здійснюється через набір класів, з яких створюється керований код. Цей набір класів називається інтерфейсом керованих класів GDI+ та складається з таких просторів імен:

- System.Drawing – доступ до основних графічних функцій GDI+;
- System.Drawing.Drawing2D – розширений набір засобів для створення двовимірної і векторної графіки;
- System.Drawing.Imaging – розширений набір графічних функцій GDI+;
- System.Drawing.Text – розширений набір функцій друкарень GDI+;
- System.Drawing.Printing – служби, пов'язані з друком;

TextRenderer – функції для рисування тексту GDI і управління його параметрами.

### 3.2. Методи рисування за допомогою графічних об'єктів

Клас Graphics інтерфейсу GDI+ містить такі методи для рисування елементів: DrawLine (прямі лінії), DrawRectangle (прямокутники), DrawEllipse (еліпси), DrawPolygon (багатокутники), DrawArc (дуги), DrawCurve (фундаментальні сплайни) та DrawBezier (сплайни Безьє). Кожен з цих методів *переобтяжений*, а це означає, що кожен метод може отримувати різні набори параметрів. Наприклад, один варіант методу DrawLine отримує об'єкт Pen і чотири цілі числа, а інший варіант методу DrawLine (з такою ж назвою) отримує об'єкт Pen та два об'єкти Point.

Крім перерахованих раніше методів для рисування ліній, прямокутників і сплайнів Безьє існують допоміжні методи, що виконують рисування декількох подібних елементів (ліній, прямокутників або сплайнів) за один виклик: DrawLines, DrawRectangles та DrawBeziers. Для методу DrawCurve також існує допоміжний метод DrawClosedCurve, що замикає криву шляхом з'єднання останньої точки кривої з першою.

Всі призначені для рисування методи класу Graphics використовують об'єкт Pen. Щоб нарисувати який-небудь елемент, потрібно створити як мінімум два об'єкти: об'єкт Graphics і об'єкт Pen. Об'єкт Pen призначений для зберігання таких атрибутів рисованого елемента, як ширина лінії і колір. Об'єкт Pen передається в кожен метод рисування як один із аргументів:

```
invoke GdipCreateFromHDC,hdc,addr graphics  
invoke GdipCreatePen1,07F000FFh,\;7F - прозорість, RGB  
rWidth,UnitPixel,addr pen
```

### 3.3. Ініціалізація та завершення

Перш ніж почати використовувати класи і функції Gdi+, необхідно ініціалізувати цю бібліотеку. Для цього на початку програми потрібно помістити виклик функції GdiplusStartup:

```
Status GdiplusStartup( ULONG_PTR* token,  
const GdiplusStartupInput* input,  
GdiplusStartupOutput* output )
```

Поля структури GdiplusStartupInput управляють різними аспектами ініціалізації: зокрема, можна задати функцію, яка викликатиметься при



виникненні помилок або перехоплюватиме всі звернення до функцій Gdi+. Конструктор за умовчанням виконує ініціалізацію структури GdiplusStartupInput з параметрами, достатніми для більшості випадків. При цьому як вихідний параметр output можна задати NULL. Для masm фрагмент коду може бути таким:

```
GdiplusStartupInput STRUCT
    GdiplusVersion DWORD ?
    DebugEventCallback DWORD ?
    SuppressBackgroundThread DWORD ?
    SuppressExternalCodecs DWORD ?
GdiplusStartupInput ENDS

...
    LOCAL gdiplusStartupInput:GdiplusStartupInput
    LOCAL gdiplusToken:DWORD
mov gdiplusStartupInput.GdiplusVersion,1 ; ініціалізація бібліотеки
and gdiplusStartupInput.DebugEventCallback,0
and gdiplusStartupInput.SuppressBackgroundThread,0
and gdiplusStartupInput.SuppressExternalCodecs,0
invoke GdiplusStartup,addr gdiplusToken,addr gdiplusStartupInput,0
```

Вихідний параметр token необхідно зберегти.

Для завершення роботи з бібліотекою необхідно викликати функцію GdiplusShutdown:

```
VOID GdiplusShutdown(
    ULONG_PTR token );
```

Тут як параметр необхідно передати те саме число, яке повернула функція GdiplusStartup в параметрі знак.

### 3.4. Клас Graphics

Конкретний об'єкт – представник класу Graphics надається у вигляді посилання методами-обробниками подій або створюється в ході виконання ряду методів відносно конкретних об'єктів, що володіють "поверхнями рисування" (клієнтська ділянка форми, кнопки, панелі, бітова матриця).

Список членів класу Graphics наведено в табл. 3.1.

Таблиця 3.1 – Список класу Graphics

Список членів класу Graphics	Призначення
Clip	Отримує або задає об'єкт Region, що обмежує ділянку рисування даного об'єкта Graphics

Закінчення табл. 3.1

ClipBounds	Отримує структуру RectangleF, яка містить у собі вирізану ділянку даного об'єкта Graphics
Compositing-Mode	Набуває значення, яке задає порядок рисування складних зображень в об'єкті Graphics
CompositingQuality	Отримує або задає якість відображення складних зображень, які виводяться в об'єкті Graphics
DpiX	Отримує горизонтальний дозвіл об'єкта Graphics
DpiY	Отримує вертикальний дозвіл об'єкта Graphics
InterpolationMode	Отримує або задає режим вставки, пов'язаний з об'єктом Graphics
IsClipEmpty	Набуває значення, яке вказує, чи є вирізана ділянку даного об'єкта Graphics порожньою
IsVisibleClipEmpty	Набуває значення, яке вказує, чи є видима вирізана ділянку даного об'єкта Graphics порожньою
PageScale	Отримує або задає масштабування між універсальними одиницями і одиницями сторінки для даного об'єкта Graphics
PageUnit	Отримує або задає одиницю вимірювання для координат сторінки даного об'єкта Graphics
PixelOffsetMode	Отримує або задає значення, яке задає порядок зсуву точок під час відображення даного об'єкта Graphics
RenderingOrigin	Отримує або задає початкове заповнення даного об'єкта Graphics для згладжування кольорних переходів і для штрихування
SmoothingMode	Отримує або задає якість заповнення для даного об'єкта Graphics
TextContrast	Отримує або задає значення корекції яскравості для відображення тексту
TextRenderingHint	Отримує або задає режим заповнення для тексту, пов'язаного з даним об'єктом Graphics
Transform	Отримує або задає універсальне перетворення для даного об'єкта Graphics
VisibleClipBounds	Отримує або задає робочий прямокутник видимої вирізаної ділянку об'єкта Graphics

Відкриті методи класу Graphics наведено в табл. 3.2.

Таблиця 3.2 – Відкриті методи класу Graphics

Назва методів класу Graphics	Призначення
AddMetafile-Comment	Додає коментар до потокового об'єкта Metafile
BeginContainer	Переобтяжений. Зберігає графічний контейнер, що містить потоковий стан даного об'єкта Graphics, а потім відкриває і використовує новий графічний контейнер
Clear	Очищає всю поверхню для рисування і виконує заливку поверхні вказаним кольором фону
Create Obj Ref	Створює об'єкт, який містить всю необхідну інформацію для створення проксі-сервера при комунікації з видаленими об'єктами
Dispose	Звільняє всі ресурси, використовувані даним об'єктом Graphics
DrawArc	Переобтяжений. Рисує дугу, яка є частиною еліпса, заданого парою координат, шириною і висотою
DrawBezier	Переобтяжений. Будує криву Безьє, визначувану чотирма структурами Point
DrawBeziers	Переобтяжений. Формує набір кривих Безьє з масиву структур Point
DrawClosed-Curve	Переобтяжений. Будує замкнуту фундаментальну криву, визначувану масивом структур Point
DrawCurve	Переобтяжений. Будує замкнуту фундаментальну криву через точки вказаного масиву структур Point
DrawEllipse	Переобтяжений. Формує еліпс, який визначається обмежувальним прямокутником, заданим за допомогою пари координат – ширини і висоти
DrawIcon	Переобтяжений. Формує зображення, яке подане вказаним об'єктом Icon, розташованим за вказаними координатами
DrawIcon-Unstretched	Формує зображення, подане вказаним об'єктом Icon, не масштабуючи його
DrawImage	Переобтяжений. Рисує заданий об'єкт Image у заданому місці, використовуючи початковий розмір
DrawImage-Unscaled	Переобтяжений. Рисує задане зображення, використовуючи його початковий фактичний розмір, у розташуванні, заданому парою координат

Продовження табл. 3.2

DrawLine	Переобтяжений. Проводить лінію, що сполучає дві точки, визначені парами координат
DrawLines	Переобтяжений. Формує набір сегментів лінії, які сполучають масив структур Point
DrawPath	Рисує об'єкт Graphicspath
DrawPie	Переобтяжений. Рисує сектор, визначений еліпсом, який заданий парою координат, шириною, висотою і двома радіальними лініями
DrawPolygon	Переобтяжений. Рисує багатокутник, визначуваний масивом структур Point
Draw-Rectangles	Переобтяжений. Рисує набір прямокутників, визначуваних структурою Rectangle
DrawString	Переобтяжений. Створює текстовий рядок у заданому місці з указаними об'єктами Brush і Font
EndContainer	Закриває потоковий графічний контейнер і відновлює стан даного об'єкта Graphics, який було збережено при виклику методу BeginContainer
EnumerateMetafile	Переобтяжений. Відправляє записи указанного об'єкта Metafile окремо від методу зворотного виклику, який відображає їх у заданій точці
EnumerateMetafile	Переобтяжений. Відправляє записи указанного об'єкта Metafile окремо від методу зворотного виклику, який відображає їх у заданій точці
Equals	Переобтяжений. Визначає, чи рівні два екземпляри Object
ExcludeClip	Переобтяжений. Оновлює вирізану ділянку даного об'єкта Graphics, щоб виключити з неї частину, визначену структурою Rectangle
FillClosed-Curve	Переобтяжений. Заповнює замкнуту фундаментальну криву, визначувану масивом структур Point
FillEllipse	Переобтяжений. Заповнює внутрішню частину еліпса, який визначається обмежувальним прямокутником, заданим за допомогою пари координат – ширини і висоти
FillPath	Заповнює внутрішню частину об'єкта GraphicsPath
FillPie	Переобтяжений. Заповнює внутрішню частину сектора, визначеного еліпсом, який заданий шириною, висотою і двома радіальними лініями

Продовження табл. 3.2

FillPolygon	Переобтяжений. Заповнює внутрішню частину багатокутника, визначеного масивом точок, заданих структурами Point
FillRectangle	Переобтяжений. Заповнює внутрішню частину прямокутника, який визначений парою координат, шириною і висотою
FillRectangles	Переобтяжений. Заповнює внутрішню частину набору прямокутників, визначуваного структурами Rectangle
FillRegion	Заповнює внутрішню частину об'єкта Region
Flush	Переобтяжений. Викликає примусове виконання всіх відкладених графічних операцій і негайно повертається, не чекаючи їх закінчення
FromHdc	Статичний. Переобтяжений. Створює об'єкт Graphics із указанного дескриптора для контексту пристрою
FromHdcInternal	Статичний. Внутрішній метод. Не використовується
FromHwnd	Статичний. Створює новий об'єкт Graphics з указанного дескриптора для вікна
FromHwndInternal	Статичний. Внутрішній метод. Не використовується
FromImage	Статичний. Створює новий об'єкт Graphics із заданого об'єкта Image
GetHalftonePalette	Статичний. Отримує дескриптор потокової півтонової палітри Windows
GetHashCode	Служить хеш-функцією для конкретного типу
GetHdc	Отримує дескриптор контексту пристрою, пов'язаний з даним об'єктом Graphics
GetLifetimeService	Витягує службовий об'єкт потокового терміну дії, який управляє засобами терміну дії даного екземпляра
GetNearestColor	Отримує колір, найближчий до указанної структури Color
GetType	Повертає Type потокового екземпляра
InitializeLifetimeService	Отримує службовий об'єкт терміну дії для управління засобами терміну дії даного екземпляра

Продовження табл. 3.2

IntersectClip	Переобтяжений. Оновлює вирізану ділянку об'єкта Graphics, включаючи в неї перетин потокової вирізаної ділянки і вказаної структури Rectangle
IsVisible	Переобтяжений. Указує, чи міститься точка, що задана за допомогою пари координат, у видимій вирізаній ділянці даного об'єкта Graphics
Measure-Character-Ranges	Отримує масив об'єктів Region, кожен з яких зв'язує діапазон позицій символів у рамках вказаного рядка
MeasureString	Переобтяжений. Вимірює вказаний рядок у процесі її створення за допомогою заданого об'єкта Font
Multiply-Transform	Переобтяжений. Помножує універсальне перетворення даного об'єкта Graphics на перетворення вказаного об'єкта Matrix
ReleaseHdc	Звільняє дескриптор контексту пристрою, отриманий у результаті попереднього виклику методу GetHdc даного об'єкта Graphics
ReleaseHdc-Internal	Внутрішній метод. Не використовується
ResetClip	Скидає вирізану ділянку даного об'єкта Graphics і робить її нескінченною
ResetTransform	Скидає матрицю універсального перетворення даного об'єкта Graphics і робить її одиничною матрицею
Restore	Відновлює стан даного об'єкта Graphics, повертаючи його до стану об'єкта GraphicsState
RotateTransform	Переобтяжений. Застосовує задане обертання до матриці перетворення даного об'єкта Graphics
Save	Зберігає поточний стан даного об'єкта Graphics і пов'язує збережений стан з об'єктом GraphicsState
ScaleTransform	Переобтяжений. Застосовує вказану операцію масштабування до матриці перетворення даного об'єкта Graphics шляхом її додавання до матриці перетворення об'єкта
SetClip	Переобтяжений. Задает як вирізану ділянку даного об'єкта Graphics властивість Clip вказаного об'єкта Graphics
ToString	Повертає String, який показує поточний Object

Закінчення табл. 3.2

Transform-Points	Переобтяжений. Перетворює масив точок з одного координатного простору в інший, використовуючи потокове універсальне перетворення і перетворення сторінки об'єкта Graphics
TranslateClip	Переобтяжений. Переводить вирізану ділянку об'єкта Graphics в указаному обсязі в горизонтальному і вертикальному напрямках
Translate-Transform	Переобтяжений. Додає заданий переклад до матриці перетворення даного об'єкта Graphics
Захищені методи	
Finalize	Перевизначений. Див. Object.Finalize. У мовах C# и C++ для функцій фіналізації використовується синтаксис деструкції
Member-wise-Clone	Створює неповну копію потокового Object

### 3.5. Клас GraphicsPath

GraphicsPath – клас, що надає послідовність сполучених ліній і кривих. Застосування використовують контури для відображення контурів фігур, заповнення внутрішніх ділянок фігур, створення зон відсікань. Контур може складатися з будь-якої кількості фігур (контурів). Кожна фігура або складена з послідовності ліній і кривих, або є геометричним примітивом. Початкова точка фігури – перша точка в послідовності сполучених ліній і кривих. Кінцева точка – остання точка в послідовності.

Фігура, що складається з послідовності сполучених ліній і кривих (початкова і кінцева точки можуть збігатися), є розімкненою фігурою, якщо вона не замкнута явно. Фігура може бути замкнута явно за допомогою методу CloseFigure, який замикає фігуру шляхом з'єднання кінцевої і початкової точок лінією. Фігура, що складається з геометричного примітиву, є замкнутою.

Для цілей заповнення та відсікання (наприклад, якщо контур візуалізується за допомогою методу Graphics.FillPath) всі розімкнені фігури замикаються шляхом додавання лінії від першої точки фігури до останньої. Нова фігура починається неявно, коли створюється контур або фігура замикається. Нова фігура створюється явно, коли викликається метод StartFigure.

При додаванні до контуру геометричного примітиву виконується додавання фігури, що містить геометричний примітив, а також неявно починається нова фігура. Отже, в контурі завжди існує потокова фігура.

Коли лінії і криві додаються до контуру, то додається неявна лінія, що сполучає кінцеву точку потокової фігури з початковою точкою нових ліній і кривих, щоб сформувати послідовність сполучених ліній і кривих.

У фігури є напрям, що визначає, як відрізки прямих і кривих прямують від початкової точки до кінцевої. Напрямок задається або порядком додавання ліній і кривих до фігури, або геометричним примітивом. Напрямок використовується для визначення внутрішніх зон контуру для цілей відсікання і заповнення. Список членів класу наведено в табл. 3.3.

Таблиця 3.3 – Члени класу GraphicsPath

Список членів класу GraphicsPath	Призначення
<b>Конструктор</b>	
GraphicsPath	Ініціалізує новий екземпляр класу GraphicsPath з перерахуванням FillMode із Alternate
<b>Властивості</b>	
FillMode	Отримує або задає перерахування FillMode, що визначає, як заповнюються внутрішні ділянки фігур в об'єкті GraphicsPath
PathPoints	Отримує точки в контурі
PathTypes	Отримує типи відповідних точок у масиві PathPoints
PointCount	Отримує кількість елементів у масиві PathPoints або PathTypes

Методи класу GraphicsPath наведені в табл. 3.4.

Таблиця 3.4 – Методи класу GraphicsPath

Методи класу GraphicsPath	Призначення
AddArc	Приєднує дугу еліпса до потокової фігури
AddBezier	Додає в потокову фігуру криву Безьє третього порядку
AddBeziers	Додає в потокову фігуру послідовність сполучених кривих Безьє третього порядку
AddClosedCurve	Додає замкнуту криву до даного контуру. Використовується крива фундаментального сплайна, оскільки крива проходить через усі точки масиву



Продовження табл. 3.4

AddCurve	Додає в потокову фігуру криву сплайна. Використовується крива фундаментального сплайна, оскільки крива проходить через всі точки масиву
AddEllipse	Додає еліпс до потокового шляху
AddLine	Додає відрізок прямої до об'єкта GraphicsPath
AddLines	Додає послідовність сполучених відрізків прямих у кінець об'єкта GraphicsPath
AddPath	Додає вказаний об'єкт GraphicsPath до даного контуру
AddPie	Додає контур сектора до даного контуру
AddPolygon	Додає багатокутник до даного контуру
AddRectangle	Додає прямокутник до даного контуру
AddRectangles	Додає послідовність прямокутників
AddString	Додає рядок тексту в даний шлях
ClearMarkers	Видаляє всі маркери з даного контуру
Clone	Створює точну копію даного контуру
CloseAllFigures	Замикає всі незамкнуті фігури в даному контурі і відкриває нову фігуру. Кожна незамкнута фігура замикається шляхом з'єднання її початкової і кінцевої точок лінією
CloseFigure	Замикає потокову фігуру і відкриває нову. Якщо потокова фігура містить послідовність сполучених ліній і кривих, то метод замикає її шляхом з'єднання початкової і кінцевої точок
CreateObjRef (успадковано від Marshal By RefObject )	Створює об'єкт, який містить всю необхідну інформацію для конструювання проксі-сервера, використовуваного для комунікації з видаленими об'єктами
Dispose	Звільняє всі ресурси, використовувані об'єктом GraphicsPath
Equals (успадковано від Object )	Визначає, чи рівні два екземпляри Object
Flatten	Перетворює кожен криву в даному контурі в послідовність сполучених відрізків прямих
GetBounds	Повертає прямокутник, що обмежує об'єкт GraphicsPath

Продовження табл. 3.4

GetHashCode (успадковано від Object)	Служить хэш-функцією для конкретного типу, придатний для використання в алгоритмах хешування і структурах даних, наприклад, у хеш-таблиці
GetLastPoint	Отримує останню точку масиву PathPoints об'єкта GraphicsPath
GetLifetimeService (успадковано від MarshalByRefObject)	Витягує службовий об'єкт потокового терміну дії, який управляє засобами терміну дії даного екземпляра
GetType (успадковано від Object)	Повертає Type потокового екземпляра
InitializeLifetimeService (успадковано від MarshalByRefObject)	Отримує службовий об'єкт для управління засобами терміну дії даного екземпляра
IsOutlineVisible	Указує, чи міститься певна точка всередині (під) контуру об'єкта GraphicsPath при відображенні його за допомогою вказаного об'єкта Pen
IsVisible	Визначає, чи міститься вказана точка в об'єкті GraphicsPath
Reset	Очищає масиви PathPoints і PathTypes і встановлює FillMode в Alternate
Reverse	Змінює порядок точок у масиві PathPoints об'єкта GraphicsPath на протилежний
SetMarkers	Установлює маркер на об'єкті GraphicsPath
StartFigure	Відкриває нову фігуру, не замикаючи при цьому потокову. Всі подальші точки, що додаються до контуру, додаються до нової фігури
ToString (успадковано від Object)	Повертає String, який подає потоковий Object
Transform	Застосовує матрицю перетворення до об'єкта GraphicsPath
Warp	Застосовує перетворення перекошу, визначуване прямокутником і паралелограмом, до об'єкта GraphicsPath

Закінчення табл. 3.2

Захищені методи	
Finalize	Перевизначений. Див. Object.Finalize.
Memberwise-Clone (успадковано від Object)	Створює неповну копію потокового Object

### 3.6. Клас Region

Клас Region (Область) призначається для створення об'єктів, які описують внутрішню частину графічної форми з прямокутників і фігур, складених із замкнутих ліній. Цей клас не успадковується.

Область є такою, що масштабується. Застосунок може використовувати ділянки для фіксації вихідних даних операцій рисування. Диспетчер вікон застосовує області для визначення ділянки зображення вікон. Ці ділянки називаються вирізаними. Застосунок може також використовувати ділянки в операціях перевірки наявності даних, наприклад, перетини точки або прямокутника з ділянкою. Застосунок може заповнювати ділянку за допомогою об'єкта Brush.

Безліч пікселів, що входять до складу регіону, може складатися з декількох несуміжних ділянок.

Методи класу наведені в табл. 3.5-3.6.

Таблиця 3.5 – Відкриті методи класу Region

Методи класу Region	Призначення
Clone	Створює точну копію об'єкта Region
Complement	Оновлює об'єкт Region, щоб включити частину вказаної структури RectangleF, не перетинну з об'єктом Region
CreateObjRef (унаследовано от MarshalByRefObject)	Створює об'єкт, який містить всю необхідну інформацію для створення проксі-сервера, використовуваного для комунікації з видаленими об'єктами
DisposeEquals	Звільняє всі ресурси, використовувані об'єктом Region
Exclude	Оновлює об'єкт Region, щоб включити частину його внутрішньої частини, не перетинну з вказаною структурою Rectangle
FromHrgn	Ініціалізував новий об'єкт Region з дескриптора вказаної існуючої ділянки GDI

Продовження табл. 3.5

GetBounds	Повертає структуру RectangleF, яка подає прямокутник, що обмежує об'єкт Region на поверхні рисунка об'єкта Graphics
GetHashCode (успадковано від Object)	Служить хеш-функцією для конкретного типу, придатний для використання в алгоритмах хешування і структурах даних
GetHrgn	Повертає дескриптор Windows для об'єкта Region в указаному графічному контексті
GetLifetimeService (успадковано від MarshalByRefObject )	Витягує службовий об'єкт потокового терміну дії, який управляє засобами терміну дії даного екземпляра
GetRegionData	Повертає об'єкт RegionData, який подає дані, що описують об'єкт Region
GetRegionScans	Повертає масив структур RectangleF, що апроксимують об'єкт Region
GetType (успадковано від Object )	Повертає Type потокового екземпляра
InitializeLifetimeService (успадковано від MarshalByRefObject )	Отримує службовий об'єкт терміну дії для управління засобами терміну дії даного екземпляра
Intersect	Замінює об'єкт Region на його перетин з указаним об'єктом Region
IsEmpty	Перевіряє чи має об'єкт Region порожню внутрішню частину на вказаній поверхні рисунка
IsInfinite	Перевіряє чи має об'єкт Region порожню внутрішню частину на вказаній поверхні рисунка
IsVisible	Перевіряє чи міститься вказаний прямокутник в об'єкті Region
MakeEmpty	Ініціалізував об'єкт Region для порожньої внутрішньої частини
MakeInfinite	Ініціалізував об'єкт Region для нескінченної внутрішньої частини
ToString (успадковано від Object )	Повертає String, який подає потоковий Object
Transform	Перетворить цей об'єкт Region за допомогою вказаного об'єкта Matrix
Translate	Зміщує координати об'єкта Region на вказану величину

Закінчення табл. 3.5

Union	Замінює об'єкт Region на його об'єднання з указаним об'єктом GraphicsPath
Xor	Замінює об'єкт Region на різницю об'єднання і його перетину з указаним об'єктом GraphicsPath

Таблиця 3.6 – Захищені методи класу Region

Методи класу Region	Призначення
Finalize	Перевизначений. Див. Object.Finalize. У мовах C# и C++ для функцій фіналізації використовується синтаксис деструкції
MemberwiseClone (успадковано від Object)	Створює неповну копію потокового Object

### 3.7. Власні елементи управління

Відомі підходи до розробки нових елементів управління:

- об'єднання стандартних елементів управління в групи (складові елементи управління);
- оголошення нових класів, що успадковують від існуючих елементів управління;
- написання нових елементів "з нуля".

Розробка складових елементів управління припускає оголошення класу, похідного від класу UserControl і використання Майстра UserControl для додавання вкладених елементів управління з подальшою настройкою створювальних елементів.

Новий елемент управління може бути побудований на основі класу – спадкоємця якого-небудь із існуючих елементів управління. У цьому випадку в новому класі вдається частково використовувати функціональності раніше оголошеного класу, можливо, зберігаючи при цьому зовнішній вигляд елемента. Наприклад, можна оголосити власний варіант класу кнопки, який успадковуватиме клас Button.

Написання нового елемента "з нуля" відрізняється від попереднього варіанта розробки вибором базового класу. У цьому випадку ґрунтуються на класі Control, який не надає нащадкам навіть елементарного графічного інтерфейсу. Процес візуалізації в цьому випадку забезпечується обробником події Paint, що перевизначається. При цьому перевизначається віртуальний метод базового класу OnPaint з одним аргументом типу PaintEventArgs, який містить інформацію про клієнтську ділянку елемента управління. Член цього класу – об'єкт типу Graphics –

забезпечує формування подання елемента управління. Другий член класу – об'єкт типу ClipRectangle – описує доступну клієнтську ділянку елемента управління.

Слід зазначити, що між двома останніми способами визначення елементів управління не існує чітких меж. В обох випадках підставою для класифікації виявляється обсяг роботи з довизначення і перевизначення методів і властивостей новостворюваного класу елементів управління.

Об'єкти класу ImageList призначаються для збереження рисунків, які можуть відобразитися іншими елементами управління. У загальному випадку цей компонент дозволяє написати код для уніфікованого каталога рисунків. До кожного рисунка можна дістати доступ за допомогою значення індексу цього рисунка. Рисунки, що відображаються, мають один і той же формат і розмір, що встановлюється у властивості ImageSize. Таким чином, на основі даної властивості може бути реалізований ефект масштабування елемента управління в сенсі зміни його видимих розмірів у разі зміни клієнтських розмірів форми.

### 3.8. Створення геометричних фігур

**Програма 3.1.** Рисування лінії, прямокутника та еліпса:

```
title рисування лінії, прямокутника та еліпса
.686                ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант ...
include \masm32\macros\macros.asm
uselib user32,gdiplus,kernel32,gdi32
```

```
GdiplusStartupInput STRUCT
    GdiplusVersion DWORD ?
    DebugEventCallback DWORD ?
    SuppressBackgroundThread DWORD ?
    SuppressExternalCodecs DWORD ?
GdiplusStartupInput ENDS
```

```
UnitPixel equ 2 ; const
```

```
.data
```

```
szAppName db "Work with GDI+",0
szClassName db "GdipExample",0
rWidth REAL4 15.0
rX1 REAL4 10.0 ; X-координата початку лінії
```

```

rY1 REAL4 10.0      ; Y-координата початку лінії
rX2 REAL4 450.0    ; X-координата закінчення лінії
rY2 REAL4 300.0    ; Y-координата закінчення лінії

```

.data?

```

hInstance dd ?
hWndMain dd ?

```

.code

```

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
              CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL gdiplusStartupInput:GdiplusStartupInput
    LOCAL gdiplusToken:DWORD

```

```

    mov gdiplusStartupInput.GdiplusVersion,1      ; ініціалізація бібліотеки
    and gdiplusStartupInput.DebugEventCallback,0
    and gdiplusStartupInput.SuppressBackgroundThread,0
    and gdiplusStartupInput.SuppressExternalCodecs,0
    invoke GdiplusStartup,addr gdiplusToken,addr gdiplusStartupInput,NULL
    mov wc.cbSize,sizeof WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW
    mov wc.lpfWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,NULL
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_WINDOW+1
    mov wc.lpszMenuName,NULL
    mov wc.lpszClassName,offset szClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov wc.hIcon,eax
    mov wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax

```

```

    invoke RegisterClassEx,addr wc

```

```

    invoke CreateWindowEx,WS_EX_APPWINDOW,addr szClassName, \
    addr szAppName, WS_TILEDWINDOW,\
    CW_USEDEFAULT,CW_USEDEFAULT, 500, 350, 0, 0, hInst, 0
    mov hWndMain,eax
    invoke ShowWindow,hWndMain,SW_SHOWNORMAL
    invoke UpdateWindow,hWndMain

```

```

    .while TRUE
        invoke GetMessage,addr msg,0,0,0
    .break .if (!eax)
        invoke TranslateMessage,addr msg
        invoke DispatchMessage,addr msg
    .endw
invoke GdiplusShutdown,addr gdiplusToken ; завершення роботи з бібл.
    mov eax,msg.wParam
    ret
WinMain endp

```

```

WndProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,\
    lParam:DWORD
    LOCAL hdc:DWORD ; резервування стека під контекст
    LOCAL ps:PAINTSTRUCT
    LOCAL graphics:DWORD
    LOCAL pen:DWORD
    .if uMsg==WM_CREATE
;push hWnd
;pop hWndMain

```

```

.elseif uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
    invoke BeginPaint,hWnd,addr ps ; виклик підготовчої процедури
    mov hdc,ecx ; збереження контексту
invoke GdiCreateFromHDC,hdc,addr graphics; створення контексту
invoke GdiCreatePen1,05F000FFh,\ ; 7F – прозорість, RGB
    rWidth,UnitPixel,addr pen
    invoke GdiDrawLine,graphics,pen,rX1,rY1,rX2,rY2
    invoke GdiDrawRectangle,graphics,pen,rX1,rY1,rX2,rY2
    invoke GdiDrawEllipse,graphics,pen,rX1,rY1,rX2,rY2

    invoke GdiDeletePen,pen
    invoke GdiDeleteGraphics,graphics
    invoke EndPaint,hWnd,addr ps

```

```

.elseif uMsg==WM_DESTROY ; якщо є повідомлення про знищення
invoke PostQuitMessage,0 ; передача повідомлення про знищення
    .else
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
    .endif ; закінчення логічної структури .IF – .ELSEIF
xor eax,ecx
    ret ; повернення з процедури
WndProc endp ; закінчення процедури

```



**start:**

```
invoke GetModuleHandle, NULL; отримання дескриптора програми
mov hInstance,eax ; збереження дескриптора програми
invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
```

```
invoke ExitProcess,0 ; повернення управління та вивільнення ресурсів
end start
```

Результат виконання програми наведено на рис. 3.1.

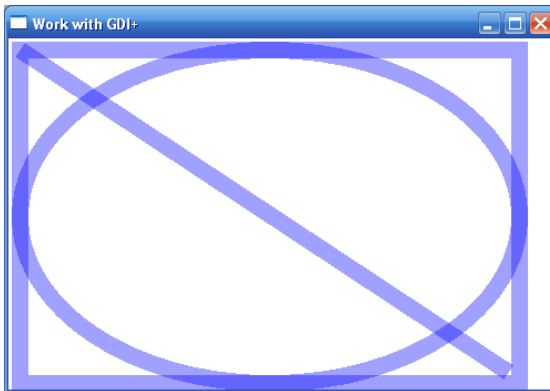


Рис. 3.1. Результат роботи програми

**Програма 3.2.** Рисування 18-кутової фігури з точками на колі:

```
.686
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\macros\macros.asm
uselib user32,gdiplus,kernel32,gdi32
```

```
GdiplusStartupInput STRUCT
    GdiplusVersion DWORD ?
    DebugEventCallback DWORD ?
    SuppressBackgroundThread DWORD ?
    SuppressExternalCodecs DWORD ?
GdiplusStartupInput ENDS
```

```
.const
    IDM_EXIT equ 2
```

```
IDM_ABOUT equ 3
IDI_ICON equ 22
UnitPixel equ 2
```

```
.data
ClassName db "SimpleWinClass",0
AppName db "Курсовой с использованием GDI+ Double buffer",0
MenuName db "FirstMenu",0
About_string db "Програму написал студент гр. КИТ-30 Поддубный Д.В.",0
wTop dd ? ; відступ зверху
wLeft dd ? ; відступ знизу
wH dd 500 ; висота вікна
wW dd 600 ; ширина вікна
cRect RECT <> ; область вікна
;GDI+
memGraphics dd 0
hBitmap dd 0
rWidth dd 1.0 ; ширина щітки в пікселях
Yc tbyte ? ; координати відносно центра
Xc tbyte ? ; координати відносно центра
cR tbyte 200.0 ; радіус
dPoints_count dd 18 ; кількість точок
cColor dd 0FF0000FFh ; перший байт повинен бути FF, останні 3 – R,G,B
tmp dd ? ; тимчасова змінна
x2 dword 2 ; x*2
dPoints qword 100 dup(0) ; масив точок (max = 100)
pushre macro
push eax
push ebx
push ecx
push edx
push esi
push edi
endm

popre macro
pop edi
pop esi
pop edx
pop ecx
pop ebx
pop eax
endm

.data?
hInstance HINSTANCE ?
```

## CommandLine LPSTR ?

```
.code
CalcPoints proc base_addr:dword
    LOCAL i:dword
    pushre
    ; Xi = Xc + R*cos(2*pi*i/n)
    ; Yi = Yc + R*sin(2*pi*i/n)
    finit
    mov edi,base_addr
    mov eax,dPoints_count
    inc eax
    mov ebx,1
calc_points:
    mov dword ptr[i],ebx

; X[i]
    fld tbyte ptr[Xc]
    fistp dword ptr[tmp]      ; tmp = Xc
    fld dword ptr[dPoints_count] ; st(0) = N
    fdivr dword ptr [i]      ; st(0) = index/N
    fldpi
    fmul                      ; st(0) = (index/N)*pi
    fimul dword ptr[x2]      ; st(0) = (index/N)*pi*2
    fcos                      ; st(0) = cos((index/N)*pi*2)
    fld tbyte ptr[cR]
    fmul                      ; st(0) = R*cos((index/N)*pi*2)
    fld tbyte ptr[Xc]
    fadd st(0),st(1)         ; st(0) = Xc+R*cos((index/N)*pi*2)
    ffree st(1)
    fistp dword ptr[tmp]
                                ; put into array (first X)

    lea esi,tmp
    mov ecx,4
    rep movsb                  ; write 2x2 words into edi [base_addr]

; Y[i]
    fld tbyte ptr[Yc]
    fistp dword ptr[tmp]      ; tmp = Yc
    fld dword ptr[dPoints_count] ; st(0) = N
    fdivr dword ptr [i]      ; st(0) = index/N
    fldpi
    fmul                      ; st(0) = (index/N)*pi
    fimul dword ptr[x2]      ; st(0) = (index/N)*pi*2
    fsin                      ; st(0) = sin((index/N)*pi*2)
    fld tbyte ptr[cR]
    fmul                      ; st(0) = R*sin((index/N)*pi*2)
```

```

fld tbyte ptr[Yc]
fadd st(0),st(1)          ; st(0) = Yc+R*sin((index/N)*pi*2)
ffree st(1)
fistp dword ptr[tmp]
                          ; put into array (second Y)

lea esi,tmp
mov ecx,4
rep movsb                ; write 2x2 words into edi [base_addr]

inc ebx
cmp eax,ebx
jne calc_points
popre
ret
CalcPoints endp

```

### **CalcRect proc h:HWND**

```

finit
invoke GetClientRect, h, addr cRect
fld dword ptr[cRect.right]
fidiv dword ptr[x2]
fstp tbyte ptr[Xc]
ffree st(0)
fld dword ptr[cRect.bottom]
fidiv dword ptr[x2]
fstp tbyte ptr[Yc]
ffree st(0)
invoke InvalidateRgn, h, NULL, FALSE
ret
CalcRect endp

```

### **PaintFigureDoubleBuffered proc hwnd:DWORD**

```

LOCAL hdc:HDC
LOCAL hGraphics:DWORD
LOCAL pen:DWORD      ; дескриптор щітки
LOCAL dWidth:DWORD
LOCAL dX1:DWORD
LOCAL dY1:DWORD
LOCAL dX2:DWORD
LOCAL dY2:DWORD
LOCAL gdiplusStartupInput:GdiplusStartupInput
LOCAL gdiplusToken:DWORD

mov gdiplusStartupInput.GdiplusVersion,1
and gdiplusStartupInput.DebugEventCallback,0
and gdiplusStartupInput.SuppressBackgroundThread,0

```

```

and gdiplusStartupInput.SuppressExternalCodecs,0

invoke GetDC, hwnd
mov hdc, eax
invoke GdiplusStartup,addr gdiplusToken,addr gdiplusStartupInput,0 ;
ініціалізація
invoke GdiplusCreateFromHDC,hdc,addr hGraphics ;
invoke GdiplusCreateBitmapFromGraphics, dword ptr[cRect.right],dword
ptr[cRect.bottom], hGraphics,addr hBitmap ; бітман
invoke GdiplusGetImageGraphicsContext, hBitmap,addr memGraphics
invoke GdiplusSetSmoothingMode, memGraphics, 4 ; тип згладжування
invoke GdiplusGraphicsClear , memGraphics, 0xffffffff ; очищаємо memGraphics

m2m dWidth,rWidth
invoke GdiplusCreatePen1,cColor,dWidth,UnitPixel,addr pen ; перо
mov eax,0
lea ecx,dPoints
mov esi,ecx
mov edi,esi
.REPEAT
    mov ebx,0
    mov edi,ecx
    .REPEAT
        .IF edi!=esi
            m2m dX1,[edi]
            m2m dY1,[edi+4]
            m2m dX2,[esi]
            m2m dY2,[esi+4]
            pushre
        .ENDIF
        mov edx,8
        add edi,edx
        inc ebx
    .UNTIL ebx==dPoints_count
    mov edx,8
    add esi,edx
    inc eax

    .UNTIL eax==dPoints_count
invoke GdiplusDrawImagePointRectI, hGraphics, hBitmap, 0, 0, 0, 0, dword
ptr[cRect.right],dword ptr[cRect.bottom], 2
invoke GdiplusDeletePen,pen
invoke GdiplusDeleteGraphics, memGraphics

```

```

    invoke GdiipDisposalImage, hBitmap
    invoke GdiipDeleteGraphics, hGraphics
    invoke GdiplusShutdown,addr gdiplusToken ; Вбиваємо GDI+
    ret
PaintFigureDoubleBuffered endp

```

```

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
    CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
mov  wc.cbSize,SIZEOF WNDCLASSEX ; кількість байтів структури
mov  wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка
вікна
mov  wc.lpfWndProc, OFFSET WndProc ; адреса процедури WndProc
mov  wc.cbClsExtra,NULL ; кількість байтів для структури
mov  wc.cbWndExtra,NULL ; кількість байтів для структури
    m2m hInst,wc.hInstance
    mov  wc.hbrBackground,COLOR_WINDOW+1 ; колір вікна
    mov  wc.lpszMenuName,OFFSET MenuName ; ім'я ресурсу меню
    mov  wc.lpszClassName,OFFSET ClassName ; ім'я класу
    invoke LoadIcon,hInstance,IDI_ICON ; ресурс піктограми
    mov  wc.hIcon,eax ; дескриптор піктограми
    mov  wc.hIconSm,eax ; дескриптор маленького віконця
    invoke LoadCursor,NULL,IDC_ARROW ; ресурс курсора
    mov  wc.hCursor,eax
    invoke RegisterClassEx, addr wc ; реєстрація класу вікна
    invoke GetSystemMetrics,SM_CXSCREEN ; ширина екрана в пікселях
        shr eax, 1
        mov ebx, wW
        shr ebx, 1
        sub eax, ebx
        mov wLeft, eax
    invoke GetSystemMetrics,SM_CYSCREEN ; висота екрана в пікселях
        shr eax, 1
        mov ebx, wH
        shr ebx, 1
        sub eax, ebx
        mov wTop, eax
    invoke CreateWindowEx,WS_EX_LEFT, ADDR ClassName, ADDR AppName,
WS_OVERLAPPEDWINDOW, wLeft,wTop,wW,wH, 0,0, hInst,0
        mov  hwnd,eax
; розрахунок точок і розмірів центра вікна
    invoke CalcRect,hwnd
    invoke CalcPoints,addr dPoints

```

```

invoke ShowWindow, hwnd, SW_SHOWNORMAL
invoke UpdateWindow, hwnd
.WHILE TRUE
    Invoke GetMessage, ADDR msg, NULL, 0, 0
    .BREAK .IF (!eax)
    invoke TranslateMessage, addr msg
    Invoke DispatchMessage, ADDR msg
.ENDW
mov eax, msg.wParam
ret
WinMain endp

```

```

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM

```

```

.IF uMsg==WM_DESTROY
    invoke PostQuitMessage, NULL

```

```

.ELSEIF uMsg==WM_CREATE

```

```

.ELSEIF uMsg==WM_COMMAND
    mov eax, wParam
    .IF ax==IDM_ABOUT
        invoke MessageBox, NULL, addr About_string, addr AppName, MB_OK
    .ENDIF
    .IF ax==IDM_EXIT
        invoke DestroyWindow, hWnd
    .ENDIF

```

```

.ELSEIF uMsg==WM_PAINT ; рисуємо в повідомленні WM_PAINT
invoke PaintFigureDoubleBuffered, hWnd ; рисуємо з подвійним буфером

```

```

.ELSEIF uMsg==WM_ERASEBKGD
; прибираємо ефект мерехтіння, забороняючи очищення фону вікна
mov eax, 1

```

```

.ELSEIF uMsg==WM_SIZE ; перераховуємо крапки при збільшенні розміру
invoke CalcRect, hWnd
invoke CalcPoints, addr dPoints

```

```

.ELSE
    invoke DefWindowProc, hWnd, uMsg, wParam, lParam
ret

```

```
.ENDIF
```

```
    xor esi,esi  
    xor eax,eax  
    ret  
WndProc endp
```

```
_start:
```

```
    invoke GetModuleHandle, NULL ; отримання дескриптора програми  
    mov hInstance,eax           ; збереження дескриптора програми  
    invoke GetCommandLine  
    mov CommandLine,eax  
    invoke WinMain, hInstance,NULL,CommandLine,  
SW_SHOWDEFAULT  
    invoke ExitProcess,eax  
end _start
```

Результат виконання програми наведено на рис. 3.2.

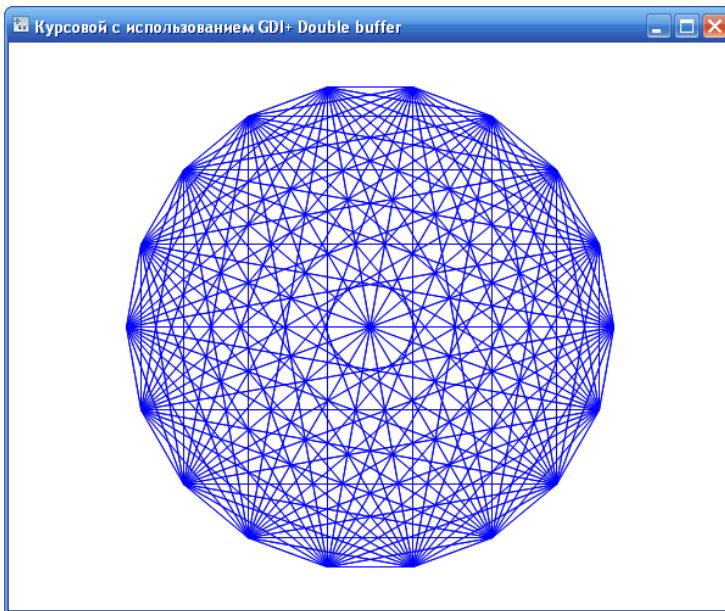


Рис. 3.2. 18-кутова фігура з точками на колі



### 3.9. Виведення тексту

Процедура оброблення повідомлень починається з оброблення WM\_CREATE.

У цьому циклі викликається функція MultiByteToWideChar, яка перетворить мультибайтові символи рядка в «широкобайтні».

Функцію викликають двічі. Перший раз для визначення кількості символів, другий – для конвертації.

Ця функція має синтаксис:

```
int MultiByteToWideChar(  
    UINT uCodePage,  
    DWORD dwFlags,  
    PCSTR pMultiByteStr,  
    int cchMultiByte,  
    PWSTR pWideCharStr,  
    int cchWideChar )
```

Параметр uCodePage задає номер кодової сторінки, пов'язаної з мультибайтовим рядком. Параметр dwFlags впливає на перетворення букв з діакритичними знаками. Зазвичай ці прапорці не використовуються, і dwFlags дорівнює 0. Параметр pMultiByteStr указує на перетворюваний рядок, а cchMultiByte визначає його довжину в байтах. Функція самостійно визначає довжину рядка, якщо cchMultiByte дорівнює -1.

Unicode – версія рядка, яка отримана в результаті перетворення, записується в буфер за адресою, вказаною в pWideCharStr. Максимальний розмір цього буфера (у символах) задається в параметрі cchWideChar. Якщо він дорівнює 0, то функція нічого не перетворює, а просто повертає розмір буфера, необхідного для збереження результату перетворення. Конверсія мультибайтового рядка в її Unicode-еквівалент проходить так:

- викликають MultiByteToWideChar, передаючи NULL у параметрі pWideCharStr та 0 в параметрі cchWideChar;

- виділяють блок пам'яті, достатній для збереження перетвореного рядка. Його розмір отримують з попереднього виклику MultiByteToWideChar;

- знову викликають MultiByteToWideChar, цього разу передаючи адресу виділеного буфера в параметрі pWideCharStr, а розмір буфера, отриманий при першому зверненні до цієї функції, – в параметрі cchWideChar.

Зворотнє перетворення виконує функція WideCharToMultiByte:

```
int WideCharToMultiByte(  
    UINT uCodePage,  
    DWORD dwFlags,  
    PCWSTR pWideCharStr,  
    int cchWideChar,  
    PSTR pMultiByteStr,  
    int cchMultiByte,  
    PCSTR pDefaultChar,  
    PBOOL pfUsedDefaultChar);
```

### Програма 3.3. Виведення тексту:

```
.686  
.model flat, stdcall  
option casemap:none ; відмінність малих та великих літер  
include \masm32\include\windows.inc  
include \masm32\include\user32.inc  
include \masm32\include\kernel32.inc  
include \masm32\include\gdiplus.inc  
includelib \masm32\lib\user32.lib  
includelib \masm32\lib\kernel32.lib  
includelib \masm32\lib\gdiplus.lib  
  
.data  
WindowTitle db "masm32",0  
ClassName db "WindowsClass",0  
szText db "GDI+ GdiDrawString RedAlfaBrush асемблер",0  
szFontName db "arial",0 ; назва шрифту  
gdiplusStartupInput dd 1  
                dd 0  
                dd 0  
                dd 0  
gdiplusToken dd 0  
fontSize dd 18.0  
layoutRect RectF <30.0, 30.0, 200.0, 200.0>  
wc WNDCLASSEX <>  
msg MSG <>  
ps PAINTSTRUCT <>  
  
.data?  
hInstance dd ?  
hWnd dd ?  
pGraphics dd ?  
pFontFamily dd ?  
pFont dd ?
```

```
pRedAlfaBrush dd ?
UnicodeText db 512 dup (?)
UnicodeFont db 32 dup (?)
```

```
.code
```

```
start:
```

```
invoke GetModuleHandle,0
mov hInstance, eax
invoke GdiplusStartup,addr gdiplusToken, addr gdiplusStartupInput,0
call WinMain
invoke GdiplusShutdown,gdiplusToken
invoke ExitProcess,0
```

```
WinMain proc
```

```
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, 0
mov wc.lpfnWndProc, offset WndProc
mov wc.cbClsExtra, 0
mov wc.cbWndExtra, 0
push hInstance
pop wc.hInstance
mov wc.hbrBackground, COLOR_BTNSHADOW
mov wc.lpszMenuName, 0
mov wc.lpszClassName, offset ClassName
```

```
invoke LoadCursor,NULL,IDC_ARROW
mov wc.hCursor,eax
```

```
invoke LoadIcon,hInstance, 0
mov wc.hIcon,eax
mov wc.hIconSm, 0
```

```
invoke RegisterClassEx,addr wc
```

```
invoke GetSystemMetrics, SM_CXSCREEN
```

```
sub eax, 400
shr eax, 1
push eax
```

```
invoke GetSystemMetrics, SM_CYSCREEN
```

```
sub eax, 200
shr eax, 1
pop edx
```

```
invoke CreateWindowEx,0,addr ClassName,addr WindowTitle,\
```

```
WS_OVERLAPPEDWINDOW or WS_VISIBLE, edx,eax,400,200, 0,0,\
hInstance,0
```

```
mov hWnd, eax
```

```
.WHILE TRUE
```

```
    invoke GetMessage, ADDR msg,NULL,0,0
```

```
    .BREAK .IF (!eax)
```

```
    invoke TranslateMessage,addr msg
```

```
    invoke DispatchMessage, ADDR msg
```

```
.ENDW
```

```
    mov eax,msg.wParam
```

```
    ret
```

```
WinMain endp
```

```
WndProc proc hWin, uMsg, wParam, lParam
```

```
    .if uMsg == WM_CREATE
```

```
        ; для перекладу символів із звичайного в широкий формат
```

```
    invoke MultiByteToWideChar,0,0,addr szText,0xffffffff,addr UnicodeText,512
```

```
    invoke MultiByteToWideChar,0,0,addr szFontName,0xffffffff,\
```

```
    addr UnicodeFont,32
```

```
    xor eax,eax
```

```
    .elseif uMsg == WM_PAINT
```

```
        invoke BeginPaint,hWin,addr ps
```

```
    invoke GdipCreateFromHDC,eax,addr pGraphics ; створення об'єкта Graphics
```

```
    invoke GdipCreateFontFamilyFromName,addr UnicodeFont, 0,\
```

```
    addr pFontFamily ; створення шрифту за іменем
```

```
    invoke GdipCreateFont,pFontFamily,fontSize,0,3,addr pFont ; шрифт
```

```
    invoke GdipDeleteFontFamily,pFontFamily
```

```
    invoke GdipCreateSolidFill,066ff0000h,addr pRedAlfaBrush ; створення кисті
```

```
    invoke GdipDrawString,pGraphics, addr UnicodeText, 0xffffffff, pFont,\
```

```
    addr layoutRect , 0, pRedAlfaBrush ; виведення тексту
```

```
    invoke GdipDeleteFont,pFont
```

```
    invoke GdipDeleteBrush,pRedAlfaBrush
```

```
    invoke GdipDeleteGraphics,pGraphics
```

```
    invoke EndPaint,hWin,addr ps
```

```
    .elseif uMsg == WM_DESTROY
```

```
        invoke PostQuitMessage, NULL
```

```
        ret
```

```
    .endif
```

```
    invoke DefWindowProc,hWin,uMsg,wParam,lParam
```

```
    ret
```

```
WndProc endp
```

```
end start
```

Результат роботи програми наведено на рис. 3.3.

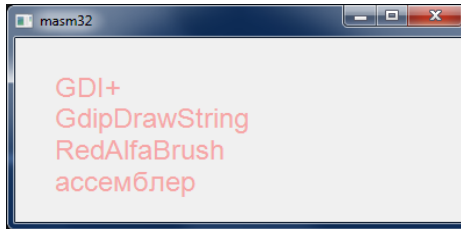


Рис. 3.3. Результат роботи програми

### Контрольні запитання

1. Які можливості має GDI+-технологія?
2. Що являють собою класи GDI+?
3. У чому полягає загальне створення програм GDI+?
4. Яку товщину в пікселях має стандартне перо та яке воно має ім'я?
5. Що означає термін переобтяженості функції?
6. Написати програму з виведенням тексту та виконанням функцій:
  - 6.1. DrawArc
  - 6.2. DrawBezier
  - 6.3. DrawBeziers
  - 6.4. DrawClosed-Curve
  - 6.5. DrawCurve
  - 6.6. DrawEllipse
  - 6.7. DrawPie
  - 6.8. DrawPolygon
  - 6.9. Draw-Rectangles
  - 6.10. FillPath

## СПИСОК ЛІТЕРАТУРИ

1. Аблязов Р.З. Программирование на ассемблере на платформе x86-64. – М.: ДМК Пресс, 2011. – 304 с.: ил.
2. Ирвин Кип. Язык ассемблера для процессоров Intel / Ирвин Кип. – 4-е изд.; пер. с англ. – М. : Издат. дом Вильямс, 2005. – 912 с.
3. Кравець В. О. Системне програмування. Програмування на мові асемблера: навч. посіб. / В.О.Кравець, О.М. Рисований. – Харків : НТУ «ХП», 2007. – 448 с.
4. Кравець В. О. Системне програмування. Асемблер під Win32 API : навч. посіб. / В. О. Кравець, О. М. Рисований. – Харків : НТУ «ХП», 2008. – 512 с.
5. Крупник А. Ассемблер. Самоучитель / А. Крупник. – С.Пб. : Питер, 2005. – 235 с.
6. Магда Ю. С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда. – С.Пб. : Питер, 2006. – 410 с.
7. Методичні вказівки до виконання та оформлення курсового проекту з курсу «Системне програмування» для студентів спеціальностей: 7.05010201 «Комп'ютерні системи та мережі», 7.05010202 «Системне програмування», 7.05010203 «Спеціалізовані комп'ютерні системи» / уклад. О. М. Рисований – Х. : НТУ «ХП», 2013. – 184 с.
8. Методичні вказівки до виконання лабораторних робіт з курсу «Системне програмування» для студентів спеціальностей: 7.05010201 «Комп'ютерні системи та мережі», 7.05010202 «Системне програмування», 7.05010203 «Спеціалізовані комп'ютерні системи» / уклад. О. М. Рисований. – Харків : НТУ «ХП», 2013. – 216 с.
9. Пирогов В. Ю. Ассемблер и дизассемблирование / В.Ю. Пирогов. – С.Пб. : БХВ-Петербург, 2006. – 464 с.
10. Рисований О. М. Цифрові пристрої і мікропроцесори. Архітектура і програмне забезпечення: навч. посіб. / О. М. Рисований, М. В. Грушенко. – Харків : ХУПС, 2005. – 384 с.
11. Рисований О. М. Системне програмування [Текст]: підручник для студентів напрямку “Комп'ютерна інженерія” вищих навчальних закладів / О. М. Рисований. – Харків : НТУ «ХП», 2010. – 912 с.
12. Рысованный А.Н. Системное программирование, Ч.1. Программирование в среде masm64 : учеб.-метод. пособие / А.Н. Рысованный. – Харьков : «Слово», 2017. – 108 с.– На рус. яз.

13. Рысованый А.Н. Системное программирование, Ч.2. Расширенные возможности программирования в среде `masm64` : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 140 с. – На рус. яз.

14. Рогожина С. І. Системне програмування та операційні системи : навч. посіб. Ч. II / С. І. Рогожина, О. М. Рисований, В. М. Федорченко. – Харків : ХУПС, 2007. – 200 с.

15. Смоленцев М.Ю. Программирование на языке Ассемблера для 32/64-разрядных микропроцессоров семейства 80x86: Учебное пособие в 3-х частях. Часть 3. – Иркутск: ИрГУПС, 2009. – 180 с.

16. Щупак Ю. А. Win32 API. Эффективная разработка приложений / Ю.А. Щупак. – С.Пб. : Питер, 2007. – 572 с.

Навчальне видання

**РИСОВАНИЙ Олександр Миколайович**

**СИСТЕМНЕ ПРОГРАМУВАННЯ**

**Графічний інтерфейс користувача (GUI)**

Навчальний посібник

для студентів спеціальностей 123 «Комп'ютерна інженерія»  
та 125 «Кібербезпека» вищих навчальних закладів

Роботу до видання рекомендував проф. *Заповольський М.Й.*

Редактор *О.С. Самініна*

План 2017 р., поз. №27

Формат 60x84 1/16. Папір офісний. Riso-друк.  
Гарнітура Times. Ум. друк. арк. 9,6.  
Наклад 500 прим. Зам. № Ціна договірна

---

Видавничий центр НТУ «ХПІ»

Свідоцтво суб'єкта видавничої справи ДК №5478 від 21.08.2017 р.

Вул. Кирпичова, 2, м. Харків, 61002