**Зміст**

[1 Загальна частина 4](#_Toc484688187)

[1.1. Постановка задачі 4](#_Toc484688188)

[1.2. Дослідження та аналіз об’єкту програмування 4](#_Toc484688189)

[1.3. Опис засобів програмної реалізації 7](#_Toc484688190)

[1.4. Вимоги до апаратного та програмного забезпечення 8](#_Toc484688191)

[2 Практична частина 9](#_Toc484688192)

[2.1. Створення та налагодження програми 9](#_Toc484688194)

[2.2. Опис програми та її алгоритмів 11](#_Toc484688195)

[2.2.1. Опис функцій системи 11](#_Toc484688196)

[2.2.2. Опис ієрархії власних класів 14](#_Toc484688197)

[2.2.3. Опис алгоритму 18](#_Toc484688198)

[2.2.4. Опис компонентів програми 19](#_Toc484688199)

[2.3. Інструкція програміста 20](#_Toc484688200)

[2.4. Інструкція оператора 20](#_Toc484688201)

[Висновок 26](#_Toc484688202)

[Список використаних джерел 27](#_Toc484688203)

**Вступ**

Графічні примітиви – мінімальні графічні об’єкти, які утворюють векторний малюнок. До графічних примітивів відносяться:

* Лінії.
* Прямокутники.
* Еліпси.
* Дуги.
* Сегменти.
* Сектори.
* криві.

Можливість малювати графічні примітиви надає графічний редактор. Він містить набір базових примітивів, завдяки котрим можна малювати більш складні фігури. Є три види графічних редакторів:

* Растрові – призначені для створення та обробки растрових зображень. Растрові, це такі зображення, які містяться у пам’яті комп’ютера як набір точок з певним кольором.
* Векторні – дозволяють користувачу створювати та редагувати векторні зображення. Векторні – це зображення, які представляються в пам’яті на математичному описанні елементарних геометричних фігур. Головним чином вони різняться від растрових тим, що при зміненні масштабу не загублюють якості, так як виражені через математичні формули.
* Гібридні – редактори, які підтримують як створення та оброблення растрової графіки, так і векторної.

На даний час існують такі відомі графічні редактори:

|  |  |  |
| --- | --- | --- |
| **Растрові** | **Векторні** | **Гібридні** |
| Photoshop | Adobe Illustrator | RasterDesk |
| Krita | Corel Draw | AutoCAD |
| Photofiltre | Inscape | Spotlight |
| GIMP |  |  |

Вище перераховані редактори займають багато пам’яті і є універсальними. Вони містять багато функцій, які не використовуються в повній мірі для створення або редагування зображення. Тому було не зайвим створити більш простий та невеликий за обсягом графічний редактор.

В даній курсовій роботі буде розглянутий векторний графічний редактор для побудови примітивів із застосуванням ООП.

Для реалізації цього проекту було використане середовище розробки Visual Studio 2015. Програма була написана на базі WinAPI (Windows application programming interfaces), на мові С++.

1. **Загальна частина**
   1. **Постановка задачі**

Основне завдання даного проекту полягає в створенні графічного редактору для малювання векторних примітивів. В програмі повинно бути передбачено два режими:

* Створення існуючих примітивів: чотирикутник, трикутник, еліпс.
* Створення користувацьких примітивів: багатокутник.

При перемиканні режиму буде змінюватись «холст».

Передбачити вікно діалогу вибору в якому можна змінити режим програми, вибрати графічного примітив, та деякі його налаштувань, такі як: товщина кордону фігури, та його колір. Також повинна бути передбачена функція зміни положення примітива на екрані із перекриванням нижче розташованих фігур. Для цього також необхідно реалізувати чергу вимальовування фігур.

Завершена програму повинна надавати такі функціональні можливості:

1. Створення примітиву через діалогове вікно

* «Вибір режиму програми» - перехід до діалогового вікна та встановлення тумблеру в режим існуючих фігур, або в режим користувацьких фігур.
* «Визначення фігури» - при режимі існуючих фігур – надається вибір фігур із існуючого списку. При режимі користувацьких фігур – вибір кількість вершин фігури.
* «Встановлення параметрів кордону фігури» - вибір кольору, та товщини фігури.
* «Реєстрація/відміна» - створення фігури, або відміна.

1. Змінення режиму роботи програми

* «Зміна режиму» - встановлення тумблеру режиму програми через діалогове вікно створення об’єкту.

1. Малювання примітиву.

* В користувацькому – режимі вказати встановити вершини мишею. Режим існуючих фігур сам малює фігуру.

1. Перетаскування примітиву

* Вибір примітиву
* Перетаскування примітиву

1. Видалення примітиву

* Вибір примітиву
* Видалення через кнопку в меню
  1. **Дослідження та аналіз об’єкту програмування**

Графічний редактор - прикладна програма (або пакет програм), що дозволяє її користувачеві створювати і редагувати зображення на екрані комп'ютера і зберігати їх в графічних форматах файлів.

Мета графічного редактору:

Створення зображень та їх оброблення і збереження у файл для використання надалі.

На сьогоднішній день існує багато різних графічних редакторів.

Типи графічних редакторів:

* Растрові – призначені для створення та обробки растрових зображень. Растрові, це такі зображення, які містяться у пам’яті комп’ютера як набір точок з певним кольором.
* Векторні – дозволяють користувачу створювати та редагувати векторні зображення. Векторні – це зображення, які представляються в пам’яті на математичному описанні елементарних геометричних фігур. Головним чином вони різняться від растрових тим, що при зміненні масштабу не загублюють якості, так як виражені через математичні формули.
* Гібридні – редактори, які підтримують як створення та оброблення растрової графіки, так і векторної.

Зараз найбільш популярними є такі графічні редактори: Adobe Photoshop, Adobe Illustrator, GIMP, Corel Draw, AutoCAD, Microsoft Paint.

Вони надають можливість створювати файл для малювання та оброблення векторних і растрових зображень. Зокрема вони надають можливість, при створенні примітивів, робити над ними певні дії такі як: зміна товщини границі об’єкту, зміна кольору границі, зміна розміру об’єкту, закрашення об’єкту кольором. Також є можливості об’єднувати примітиви у один. Зокрема деякі з них працюють по багатошаровому принципу, коли є декілька «холстів» і вони взаємодіють між собою певним заданим чином.

Adobe Photoshop – графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в області комерційних засобів редагування растрових зображень, і найвідомішим продуктом фірми Adobe. Часто цю програму називають просто Photoshop (Фотошоп). У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка була припинена, починаючи з третьої версії продукту. Для версії CS і CS6 можливий запуск під Linux за допомогою альтернативи Windows API — Wine.

Adobe Illustrator – векторний графічний редактор, розроблений фірмою Adobe Systems. Він був задуманий як редактор векторної графіки, але дизайнери використовують його для різних цілей: для реклами, листівок, плакатів, книжок, графічних романів, розкадровок, журналів, газет та іншого. Програма забезпечує користувача великою кількістю інструментів для малювання та управлянні кольорами та текcтом.

GIMP (The GNU Image Manipulation Program) – растровий графічний редактор, із деякою підтримкою векторної графіки. Проект розпочали 1995 року Spencer Kimball і Peter Mattis як навчальний проект в Берклі. В 1997, після закінчення ними університету GIMP став частиною проекту GNU. Програма підтримується та розвивається товариством добровольців, ліцензована за умовами GNU General Public License версії 3+, починаючи з релізу 2.8. Символом GIMP є койот Вілбер (Wilber). Програма працює на системах Microsoft Windows, Gnu/Linux, FreeBSD (або OpenBSD), MacOS X, OpenSolaris.

До сфер застосування GIMP належать цифрове ретушування знімків, створення цифрової графіки, комбінування й цифрова обробка зображень, автоматизовані операції над графічними файлами, перетворення файлів з одного формату в інший.

GIMP часто використовується як вільна й безкоштовна альтернатива до Adobe Photoshop.

CorelDraw – це векторний графічний редактор, який розроблений та продається компанією Corel Corporation.

У 1985 році Майкл Коупленд заснував Corel, як компанію, націлену на розробку систем для верстки друкованої продукції для комп'ютерів на основі процесорів Intel. У 1987 році компанія Corel найняла фахівців з розробки програмного забезпечення — Майкла Буіллона і Пат Бейрн для розробки програми створення векторних ілюстрацій

CorelDraw в даний час працює на Windows XP, Windows Vista, Windows 7 і Windows 8

Кілька інновацій в векторній графіці прийнято приписувати CorelDraw: інструмент редагування вузлів, який виконує різні функції залежно від об'єкта, підгонка тексту під рамку виділення, швидкий вибір кольору заливки контура, перспективна проекція, складна градієнтна заливка.

AutoCAD — дво- і тривимірна система автоматизованого проектування і креслення розроблена компанією Autodesk. Перша версія була випущена в 1982 році. AutoCAD і спеціалізовані додатки на його основі знайшли широке застосування в машинобудуванні, будівництві, архітектурі та інших галузях промисловості. Вперше випущений в грудні 1982 року AutoCAD був однією з перших програм САПР для роботи на персональних комп'ютерах, зокрема, IBM PC. У той час, більшість інших CAD-програм працювали на великих ЕОМ.

Поточна версія програми (AutoCAD 2012) включає в себе повний набір інструментів для комплексного тривимірного моделювання (підтримується твердотільне, поверхневе і полігональне моделювання). AutoCAD дозволяє отримати високоякісну візуалізацію моделей з допомогою рендеринга mental ray. Також в програмі реалізовано управління тривимірним друком (результат моделювання можна відправити на 3D-принтер) і підтримка хмар точок (дозволяє працювати з результатами 3D-сканування).

AutoCAD підтримує декілька інтерфейсів API для налаштування і автоматизації. До них відносяться AutoLISP, Visual LISP, VBA, .NET і ObjectARX, яка є C++ бібліотекою класів, яка також була базою для розширення продуктів AutoCAD в конкретних областях, для створення продуктів, таких як AutoCAD Architecture, AutoCAD Electrical, AutoCAD Civil 3D, або сторонніх AutoCAD-додатків.

* 1. **Опис засобів програмної реалізації**

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

**C++** (вимовляється Сі-плюс-плюс) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об’єктно-орієнтованої, узагальненої та процедурної.

При створенні С++ прагнули зберегти сумісність з мовою С. Більшість програм на С справно працюватимуть і з компілятором С++. С++ має синтаксис, заснований на синтаксисі С.

Нововведеннями С++ порівняно з С є:

* Підтримка об'єктно-орієнтованого програмування через класи;
* Підтримка узагальненого програмування через шаблони;
* Доповнення до стандартної бібліотеки;
* Додаткові типи даних;
* Обробка винятків;
* Простори імен;
* Вбудовані функції;
* Перевантаження операторів;
* Перевантаження імен функцій;
* Посилання і оператори управління вільно розподіленою пам'яттю.

У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java.

При створенні С++ прагнули зберегти сумісність з мовою С. Більшість програм на С справно працюватимуть і з компілятором С++. С++ має синтаксис, заснований на синтаксисі С.

Стандартна бібліотека С++ включає стандартну бібліотеку С з невеликими змінами, які роблять її відповіднішою для мови С++. Інша велика частина бібліотеки С++ заснована на Стандартній Бібліотеці Шаблонів (STL). Вона надає такі важливі інструменти, як контейнери (наприклад, вектори і списки) і ітератори (узагальнені вказівники), що надають доступ до цих контейнерів як до масивів. Крім того, STL дозволяє схожим чином працювати і з іншими типами контейнерів, наприклад, асоціативними списками, стеками, чергами.

Microsoft Visual C++ (MSVC) – інтегроване середовище розробки програмного забезпечення на мові C++, розроблена фірмою Microsoft.

Windows API (WinAPI application programming interfaces) — загальне найменування для цілого набору базових функцій інтерфейсів програмування застосунків операційних систем сімейства Windows корпорації Майкрософт. Це є найпрямішим способом взаємодії додатків з Windows.

В якості середовища програмування було обрано Microsoft Visual C++.

Це середовище підтримує мову програмування C++, на основі WinAPI, що надає усі необхідні інструменти для створення програми.

Windows API спроектований для використання в мові С для написання прикладних програм, призначених для роботи під управлінням операційної системи MS Windows. Робота через Windows API - це найбільш близький до операційної системи спосіб взаємодії з нею з прикладних програм. Більш низький рівень доступу, необхідний тільки для драйверів пристроїв, в поточних версіях Windows надається через Windows Driver Model.

Windows API представляє собою безліч функцій, структур даних і числових констант, які узгодженні з мовою С. Всі мови програмування, здатні викликати такі функції і оперувати такими типами даних в програмах, що виконуються в середовищі Windows, можуть користуватися цим API. Зокрема, це мови C ++, Pascal, Visual Basic і багато інших.

Для полегшення програмування під Windows, в компанії Microsoft і сторонніми розробниками було зроблено безліч спроб створити бібліотеки і середовища програмування, які частково або повністю приховують від програміста особливості Windows API, які можуть надати ту чи іншу частину його можливостей в більш зручному вигляді. Зокрема, сама Microsoft в свій час пропонувала бібліотеки Active Template Library (ATL) / Windows Template Library (WTL), Microsoft Foundation Classes (MFC), .Net / WinForms / WPF, TXLib. Компанія Borland (нині Embarcadero) пропонувала OWL і VCL. Є крос-платформні бібліотеки, такі як Qt, Tk і багато інших. Вагома частина цих бібліотек сконцентрована на полегшенні програмування графічного інтерфейсу користувача.

Для полегшення перенесення на інші платформи програм, написаних з опорою на Windows API, зроблена бібліотека Wine.

* 1. **Вимоги до апаратного та програмного забезпечення**

Вимоги до апаратного забезпечення:

* Оперативна пам'ять: 1,6 - 2 Мб.
* Процесор: Intel Pentium x86 і вище, частота від 300 MHz.
* Touchscreen або миша
* Відеокарта: інтегрована.
* Вимоги до програмного забезпечення:
* Операційна система: Windows 10.

1. **Практична частина**
2. 1. **Створення та налагодження програми**

Для написання програми була використана мова Visual C++ у середовищі програмування Microsoft Visual C++ пакету Microsoft Visual Studio компанії Microsoft.

Проект типу Win32 Project надає можливість створити програму, максимально приближеної до операційної системи. Цим самим програміст отримує можливість написати свою програму найоптимальнішим чином. І з якомога меншими ресурсними витратами.

Створення ключових компонентів програми:

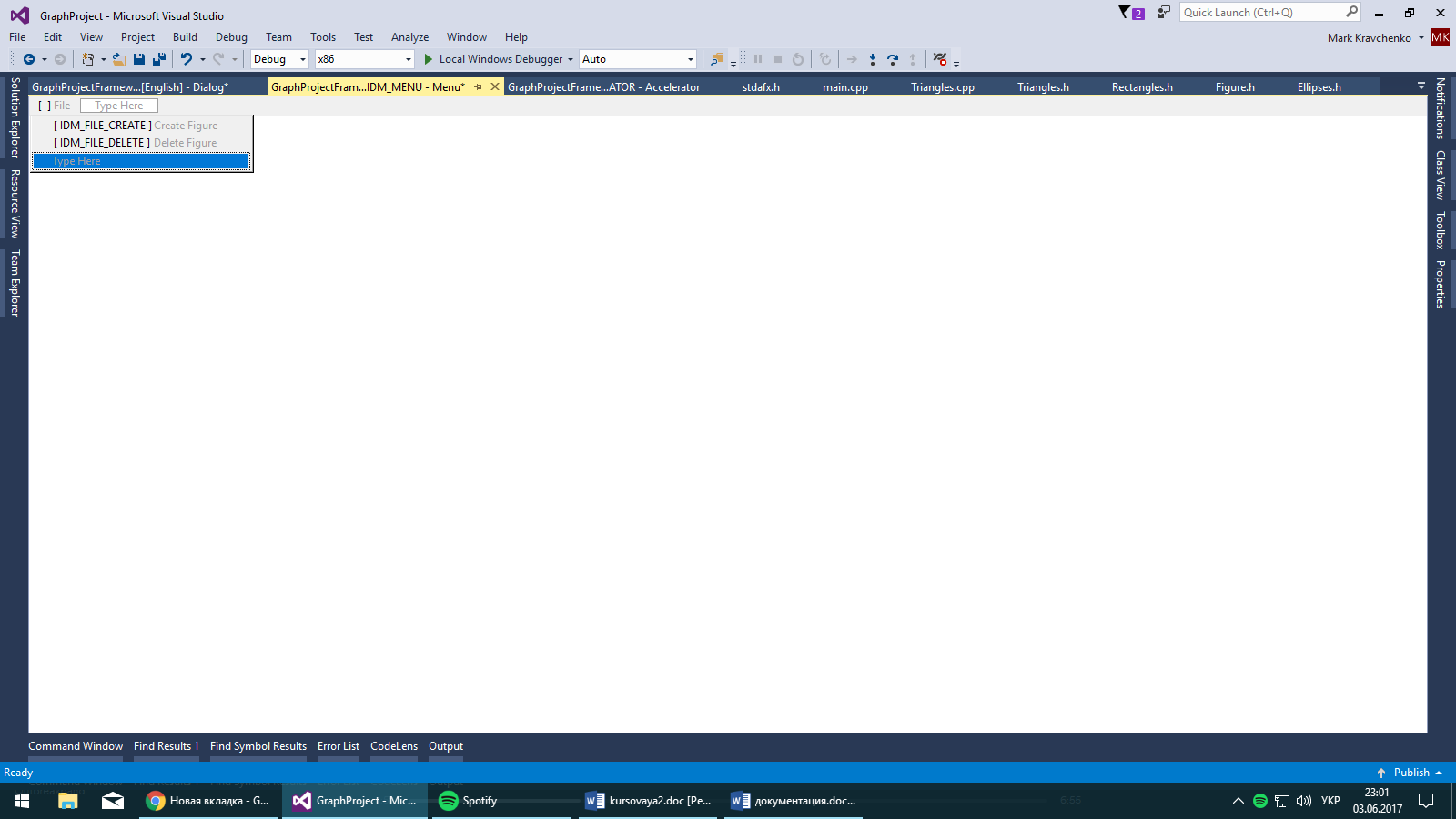
* Оголошення всіх необхідних змінних для роботи програми.
* Створення головної функції програми wWinMain. Ця функція є точкою входу для WinAPI застосування з використанням Unicode. Створення головного циклу програми у середині головної функції. Головний цикл програми при отриманні повідомлення від операційної системи пересилає його у функцію обробника подій, яка належить даному вікну, на якому виникла подія через операційну систему або користувача.
* Створення файлу ресурсів.

Цей файл містить ресурси для створення меню, діалогових вікон, таблиць акселераторів, іконки, строкові таблиці.

* Створення фалу-заголовку ресурсів.  
  В Цьому файлі зберігаються ID різних компонентів-ресурсів, які містяться у всьому проекті, та зокрема у файлі ресурсів. ID ресурсів директивою #define замінюються на відповідне значення.

Створення вікон та форм

* Створення головного меню – меню створюється за допомогою редактора ресурсів, яка включена у склад Visual Studio (див. рис. 2.1).

Рисунок 2.1 – Головне меню

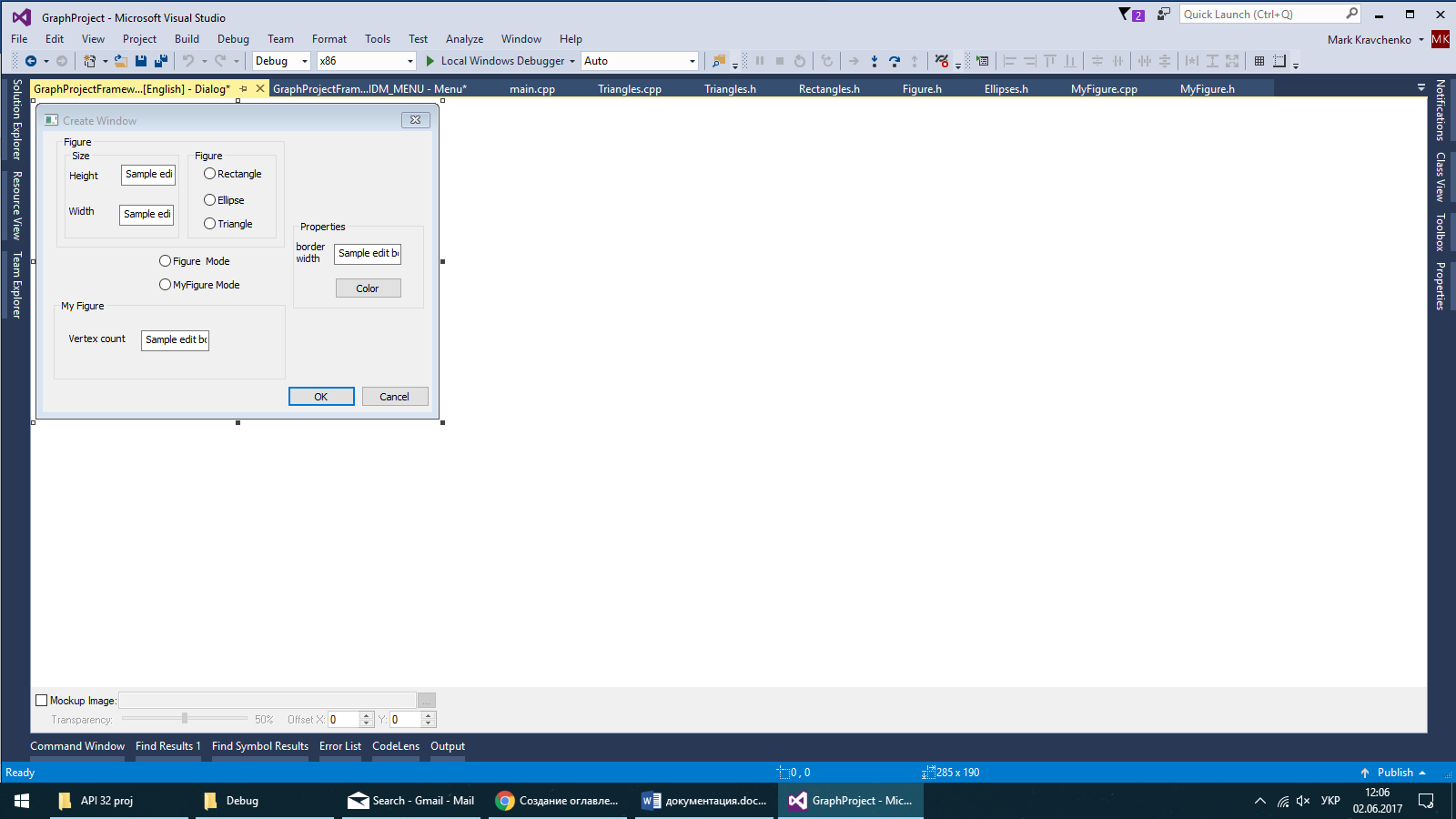
* Реєстрація класу головного вікна.
* Створення головного вікна.
* Створення діалогового вікна – вікно вибору фігури і встановлення її параметрів (див. рис. 2.2).

Рисунок 2.2 – Діалогове вікно створення фігури

* Реєстрація класу діалогового вікна вибору кольору.
* Створення діалогового вікна вибору кольору із стандартної бібліотеки “commdlg.h”.

Створення головних функцій обробки повідомлень.

* Створення функції обробки повідомлень головного вікна.
* Створення функції обробки повідомлень діалогового вікна.

Створеня та опис класів.

* Клас існуючих фігур та його дочірні класи.
* Клас користувацьких фігур.
  1. **Опис програми та її алгоритмів**

Готова програма дозволяє користувачеві створювати власні фігури просто встановивши вибрану кількість вершин на холсті, або ж вибрати із існуючих фігур. При виборі існуючої фігури програма дозволяє вказувати її розмір. Також колір і товщину рамки можна встановити як для існуючої так і для користувацької фігури. Програма ділиться на два режими: користувацьких та існуючих фігур, тому програма дозволяє переключати режими роботи.

* + 1. **Опис функцій системи**

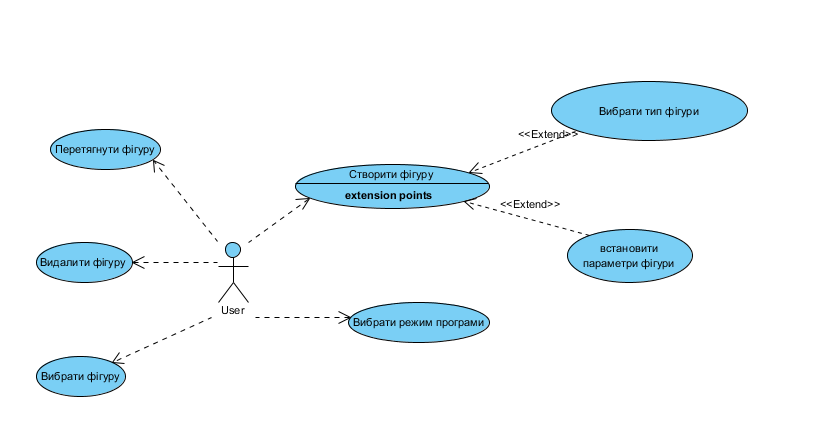
Функції системи зображені на рисунку 2.3 у вигляді UML діаграми прецедентів з одним актором – Користувачем (User).

Рисунок 2.3 – UML діаграма прецедентів

Ця діаграма показує, що користувач може:

1. Створити фігуру:

Для цього необхідно на головному меню, яке знаходиться зверху, перейти на вкладку “File”(Файл) та вибрати підпункт “Create Figure”(Створити фігуру), програма відкриє діалогове вікно вибору та налаштування фігури. На формі знаходиться 3 групових блоки і перемикач режими роботи програми.

Для того щоб створити фігуру із існуючого списку, необхідно активувати “Figure Mode”(режим існуючих фігур), потім встановити наступні параметри, які знаходяться в груповому блоці “Figure”: “Height” (висота), “Width” (ширина), та вибрати одну з даних фігур: Rectangle (чотирикутник), Ellipse (Еліпс), Triangle (Трикутник), і встановити властивості фігури у груповому блоці “Properties”(властивості). Для цього необхідно ввести значення “border width”(товщина кордону), і вибрати колір через діалогове вікно вибору кольору, натиснувши на відповідну кнопку “color”(колір), коли всі параметри встановлені необхідно натиснути кнопку “Ok” для створення фігури, або кнопку “Cancel” для виходу із діалогового вікна без створення фігури. Якщо користувач натисне кнопку “Ok” – фігура намалює себе з відповідними параметрами, що буди задані користувачем у діалоговому вікні створення фігури.

Для того щоб створити користувацьку фігуру, необхідно активувати “MyFigure Mode”(режим користувацьких фігур), далі користувачу необхідно у груповому блоці “Vertex count”(кількість вершин) – ввести кількість вершин фігури, яку він хоче створити, після того, як значення буде задано необхідно перейти до групового блоку “Properties”(параметри), та встановити ширину кордону фігури “border width”, та вибрати колір фігури за допомогою діалогового вікна вибору кольору, натиснувши кнопку “color”(колір). Після того, як всі властивості фігури будуть встановлені, необхідно або натиснути на кнопку “Ok” для створення фігури, або “Cancel” для виходу із діалогового вікна створення фігури. У випадку якщо користувач натиснув кнопку “Ok”, програма переводить користувача у режим малювання фігури, тобто користувач повинен ініціалізувати вершини фігури. Для того, щоб встановити веришини фігури, користувачу необхідно натискати на холсті програми ЛКМ(ліву кнопку миші), стільки раз, скільки вершин він вибрав у діалоговому вікні створення фігури. Після того, як всі координати відповідних вершин фігури будуть задані – фігура намалює себе.

1. Вибрати режим роботи програми.

Для цього необхідно перейти у діалогове вікно створення фігури, та вибрати режим роботи програми: “Figure mode”, “MyFigure mode”, залежно від того, користувацьких фігур чи існуючий фігур режим, користувач хоче використовувати надалі. Після цього необхідно підтвердити дію, тобто закрити діалогове вікно кнопкою “Ok”.

1. Вибрати фігуру.

Для цього необхідно перейти у діалогове вікно створення фігури, та вибрати режим роботи програми: “Figure mode”, “MyFigure mode, залежно від того, користувацьку чи існуючу фігуру, користувач хоче вибрати(див. пункт 2), після цього необхідно підтвердити дію, тобто закрити діалогове вікно кнопкою “Ok”. того . Користувачу необхідно натиснути лівою кнопкою миші на видимий регіон фігури. Фігура програмно буде переведена на верхній пласт, і цим самим вона буде програмно розглядатися як вибрана.

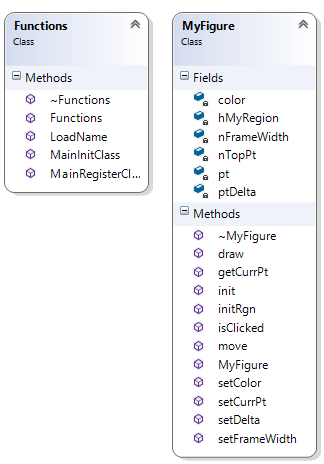
1. Видалити фігуру.

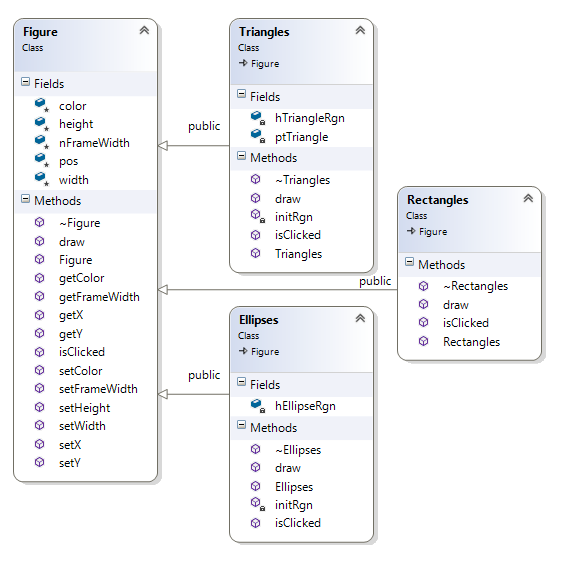
Для цього необхідно перейти у діалогове вікно створення фігури, та вибрати режим роботи програми: “Figure mode”, “MyFigure mode”, залежно від того, користувацьку чи існуючу фігуру, користувач хоче видалити, після цього необхідно підтвердити дію, тобто закрити діалогове вікно кнопкою “Ok”. Програма буде переведена у відповідний режим. Надалі необхідно вибрати фігуру, яку користувач хоче видалити, для цього необхідно натиснути на регіон, що належить фігурі, якщо вона не знаходиться зверху інших фігур   
(див. пункт 3). Для видалення фігури необхідно перейти у меню, зверху екрану, і виконати “File”-“Delete Figure”. Як тільки команда буде виконана – фігура буде видалена.

1. Користувач може Перетягнути фігуру.

Для цього користувачу необхідно вибрати режим роботи програми: “Figure mode”, “MyFigure mode, залежно від того, користувацьку чи існуючу фігуру, користувач хоче перетягнути(див. пункт 2). Потім необхідно вибрати фігуру (див. пункт 3). Після цього вибрану фігуру можна перетягувати по екрану, натиснувши ЛКМ (ліву кнопку миші), на регіоні, що належить фігурі, та тягнучи мишу по області, що належить програмі або не належить. Щоб припинити перетягування фігури – необхідно відпустити натиснуту ЛКМ.

* + 1. **Опис ієрархії власних класів**

В процесі написання графічного редактору було розроблено ієрархію класів представлену на рисунку 2.4

Рисунок 2.4 – Ієрархія класів

Опис даної ієрархії класів:

Батьківський клас “Figure” – це клас, існуючих фігур. Оскільки в програмі є три існуючих фігури: “Rectangle”, “Triangle”, “Ellipse”, то відповідно від базового класу унаслідуються три дочірніх класи. Цей клас, керує існуючими фігурами. У таблиці 2.1 представлені властивості класу “Figure”.

Таблиця 2.1 – Властивості класу “Figure”

|  |  |
| --- | --- |
| int height | висота фігури |
| int width | ширина фігури |
| int nFrameWidth | товщина кордону фігури |
| COLORREF color | Колір фігури |
| RECT pos | ліва верхня, та права нижня координати фігури |

Утаблиці 2.2 представлено опис методів класу “Figure”

Таблиця 2.2 – Методи класу “Figure”

|  |  |
| --- | --- |
| virtual void draw | малює об’єкт класу, реалізується в дочірньому класі по-різному, в залежності від того, яка фігури вибрана. |
| virtual bool isClicked | При натискані кнопки миші, ця функція перевіряє, чи належить координата миші даній фігурі. Визначається в дочірніх класах. |
| int getY | отримати Y-координату положення фігури. |
| int getX | отримати X-координату положення фігури. |
| void setY | встановити Y-координату положення фігури. |
| void setX | встановити X-координату положення фігури. |
| void setWidth | встановити довжину фігури |
| void setHeight | встановити висоту фігури |
| int getFrameWidth | отримати ширину кордону фігури |
| void setFrameWidth | встановити ширину кордону фігури |
| COLORREF getColor | отримати колір кордону фігури |
| void setColor | встановити колір кордону фігури |

Дочірній клас “Rectangle” – малює прямокутник. Властивостей данного класу не існує. Методи класу представлені в таблиці 2.3

Таблиця 2.3 – Методи класу “Rectangle”

|  |  |
| --- | --- |
| void draw | функція малює прямокутник по заданим параметрам |
| bool isClicked | функція визначає, при натисканні миші, чи попадає координата миші у даний прямокутник |

Дочірній клас “Ellipse” – малює еліпс. Властивості данного класу представлені в таблиці 2.4.

Таблиця 2.4 – Властивості класу “Ellipse”

|  |  |
| --- | --- |
| HRGN hEllipseRgn | регіон, який належить даному еліпсу |

Методи класу “Rectangle” представлені в таблиці 2.5

Таблиця 2.5 – Методи класу “ Ellipse”

|  |  |
| --- | --- |
| void draw | малює еліпс по заданим параметрам |
| bool isClicked | функція визначає, при натисканні миші, чи попадає координата миші в регіон даного еліпсу |
| void initRgn | При перетяганні фігури змінює положення регіону фігури. |

Дочірній клас “Triangle” – малює трикутник. Властивості данного класу представлені в таблиці 2.6.

Таблиця 2.6 – Властивості класу “Triangle”

|  |  |
| --- | --- |
| HRGN hTriangleRgn | регіон, який належить даному трикутнику |
| POINT ptTriangle | містить координати трьох вершин трикутника |

Методи класу “ Triangle” представлені в таблиці 2.7

Таблиця 2.7 – Методи класу “Triangle”

|  |  |
| --- | --- |
| void draw | малює трикутник по заданим параметрам |
| bool isClicked | функція визначає, при натисканні миші, чи попадає координата миші в регіон даного трикутника |
| void initRgn | При перетяганні фігури змінює положення регіону фігури. |

Також в даній ієрархії класів міститься ще один клас, “MyFigure” – цей клас відповідає за фігури користувача. Властивості даного класу представлені в таблиці 2.8

Таблиця 2.8 – Властивості класу “MyFigure”

|  |  |
| --- | --- |
| POINT pt | масив координат, який містить координати вершин користувацької фігури. |
| SIZE ptDelta | різниця між сусідніми двома координатами |
| HRGN hMyRegion; | рагіон |
| COLORREF color; | колір фігури |
| int nFrameWidth; | шиирна кордону фігури |
| int nTopPt; | кількість вершин користувацької фігури |

Методи класу “MyFigure”, які забезпечують роботу з об’єктами цього класу представлені в таблиці 2.9.

Таблиця 2.9 – Методи класу “MyFigure”

|  |  |
| --- | --- |
| void init(); | анулює необхідні параметри |
| void setCurrPt | встановлює координату заданій точці многокутника |
| POINT getCurrPt | повертає координати точки |
| void draw | малює многокутник |
| void initRgn | створює регіон многокутника |
| bool isClicked | функція визначає, при натисканні миші, чи попадає координата миші у даний прямокутник |
| void setColor | встановлює колір границі многокутника |
| void setFrameWidth | встановлює товщину кордону многокутника |
| void setDelta | встановлює різницю між координатою миші та базовою точкою многокутника |
| void move | переміщує фігуру у нову точку |

* + 1. **Опис алгоритму**

На рисунку 2.5 зображено циклічний алгоритм роботи програми.

Згідно алгоритму, програма починає роботу із завантаження головного вікна. Потім алгоритм розгалуджується для обробки кожної дії, яку програма надає користувачу.

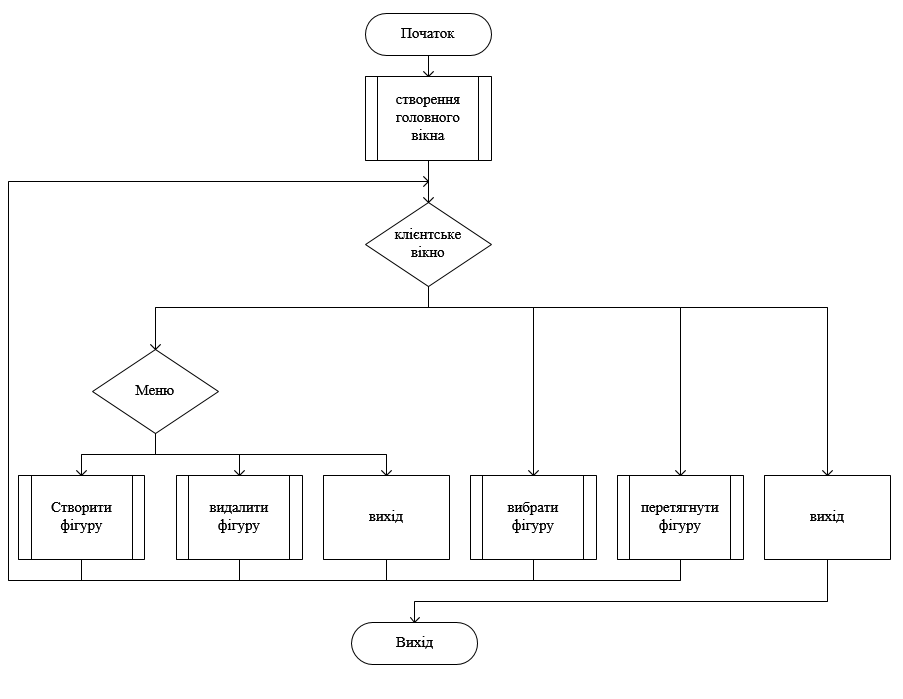
Якщо користувач обрав “Menu”-“Create Figure” – програма завантажить діалогове вікно створення фігури. Після встановлення всіх необхідних параметрів для створення фігури, програма обробить всі дані, та виконає вимальовування фігури на екран.

Якщо користувач натиснув лівою кнопкою миші на регіон клієнтської частини екрану, що належить певному многокутнику, то многокутник буде вибраний, тобто програма переведе його у вверх над усіма фігурами, і буде вважати його активним.

Якщо користувач обрав “Menu”-“Create Figure” – програма видалить вибраний многокутник, і він більше не буде виводитись на екран.

Якщо користувач вибрав фігуру і, не відпускаючи лівої кнопки миші, почав змінювати положення курсору, то програма буде перетягувати фігуру відповідно за мишею.

При натисканні на червоний хрест правого верхнього кута, то програма буде завершена.

Рисунок 2.5 – Алгоритм роботи програми

* + 1. **Опис компонентів програми**

Під час написання програми були створені наступні компоненти, які представлені в у вигляді двох файлів .h та .cpp, які зображені на рисунку 2.6

main – головний модуль, в якому створюється головне вікно, та виконується цикл отримання повідомлень від операційної системи.

WindowFunctions – ключовий модуль, в якому створюються об’єкти класів MyFigure та Figure, і реалізуються всі віконні функції оброки повідомлень операційної системи.

ObjectManage – модуль який керує виводом та створенням об’єктів многокутників, що створив користувач.

MyFigure – модуль, що містить реалізацію класу MyFigure.

Figure – модуль, що містить реалізацію класу Figure.

Rectangle – модуль, що містить реалізацію класу Rectangle.

Ellipse – модуль, що містить реалізацію класу Ellipse.

Triangle – модуль, що містить реалізацію класу Triangle.

stdafx – модуль, що підключає ключові зовнішні бібліотеки для роботи застосування.

resources – модуль, що містить користувацькі ресурси для роботи програми.

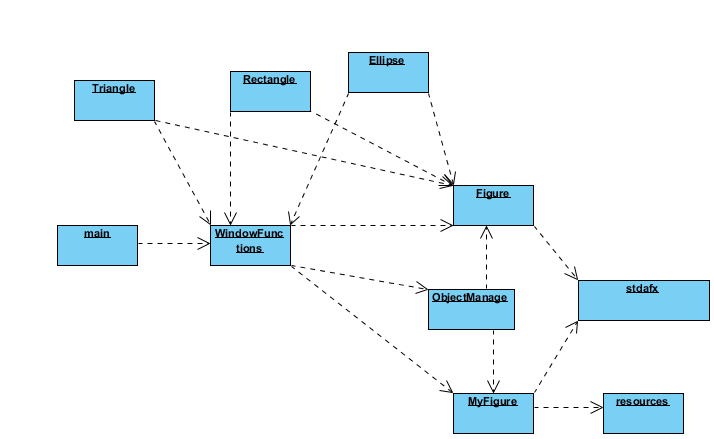


Рисунок 2.6 – Опис компонентів програми

* 1. **Інструкція програміста**

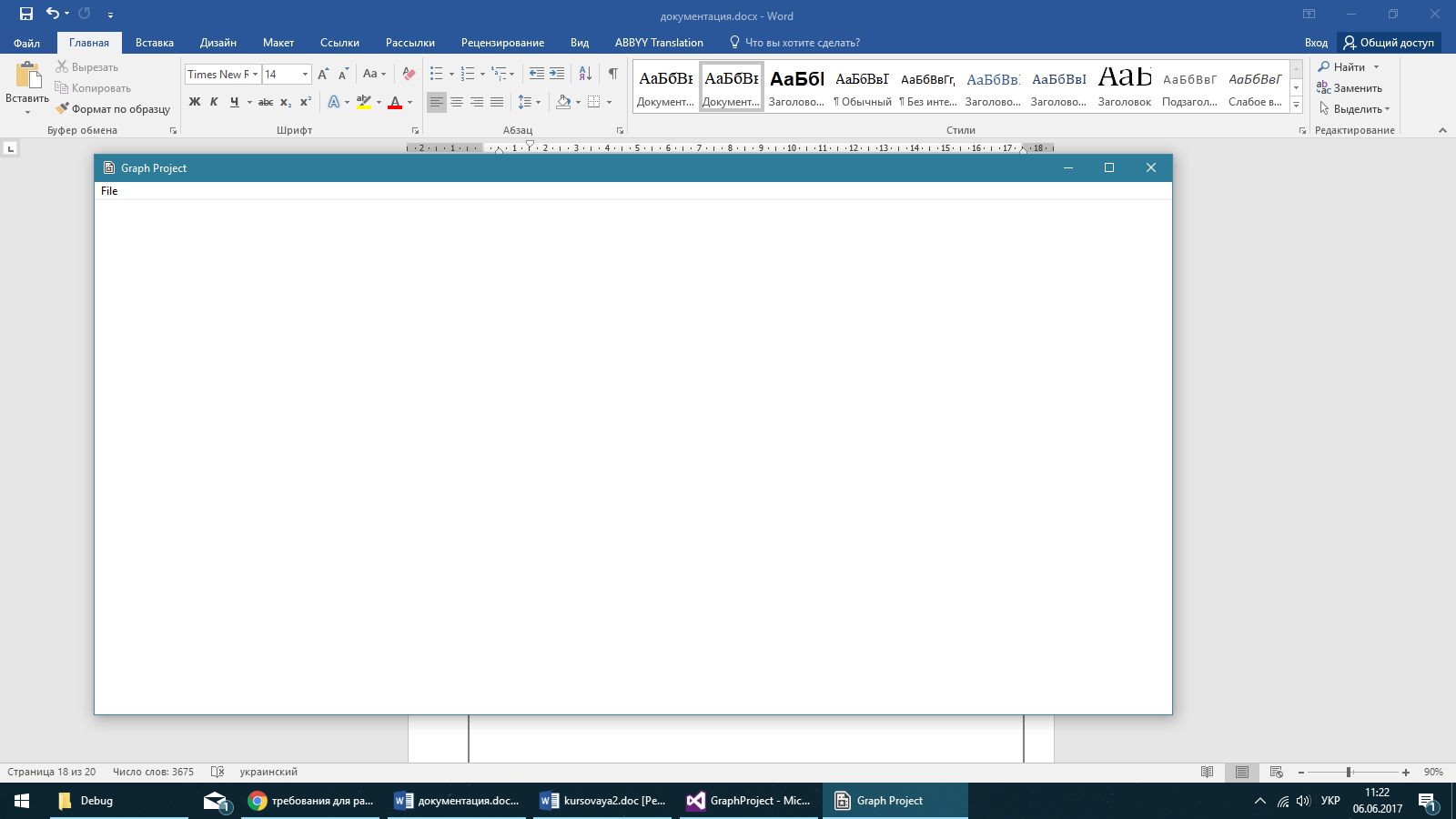
Проект являє собою графічний редактор, для пристроїв, що працюють під керуванням ОС Windows 10.

* Автор: Кравченко Марк Олександрович.
* Мова програмування: Visual C++.
* Cередовище програмуваяння: Visual Sudio 2015.
* Вимоги до апаратного забезпечення:
* Оперативна пам'ять: 16 Мб.
* Процесор: Intel Pentium x86 і вище, частота від 300 MHz.
* Touchscreen або миша.
* Клавіатура.
* Вимоги до програмного забезпечення:
* Операційна система: Windows 10.
  1. **Інструкція оператора**

Для роботи програми необхідно мати файл GraphProject.exe.

Для запуску програми необхідно натиснути два рази ЛКМ на файл GraphProject.exe

Після відкриття програми, буде створено головне вікно, що зображене на рисунку 2.7

Рисунок 2.7 – Головне вікно

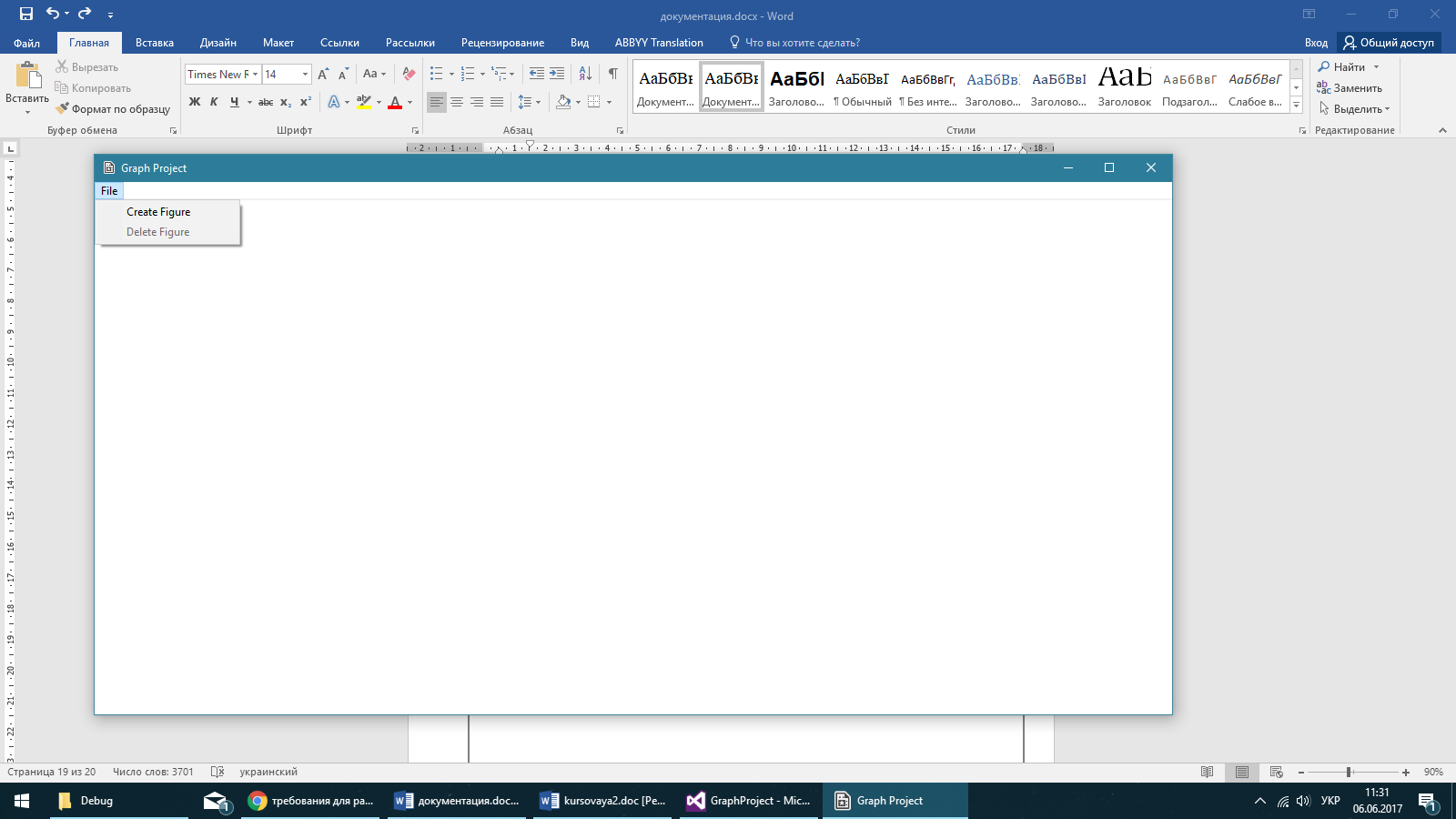
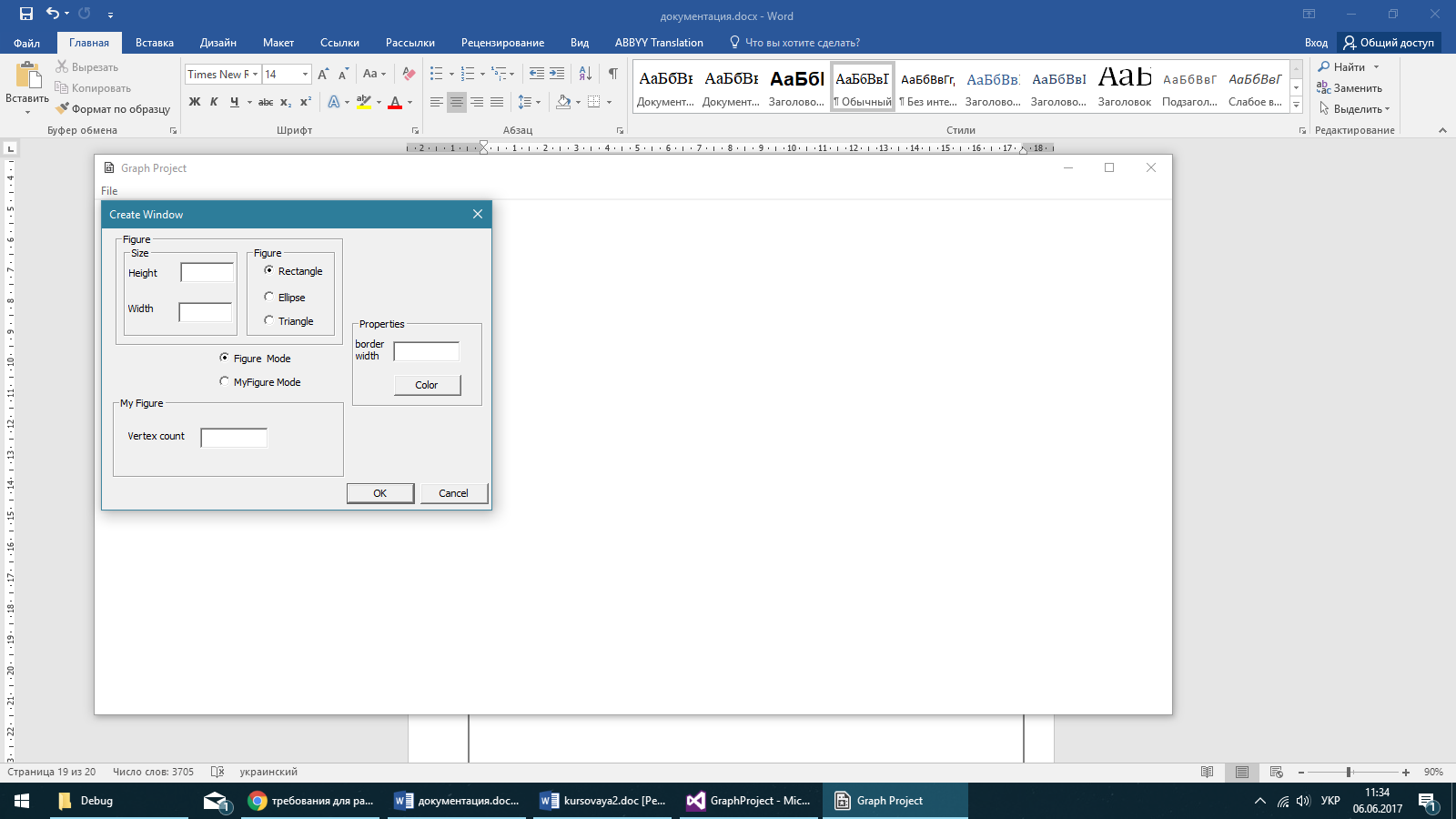
Після відкриття програми, щоб створити фігуру, користувач повинен відкрити діалогове вікно створення фігур, через меню (див. рис. 2.8), яке зображено на рисунку 2.9.

Рисунок 2.8 – Меню програми

Рисунок 2.9 – Діалогове вікно створення фігури

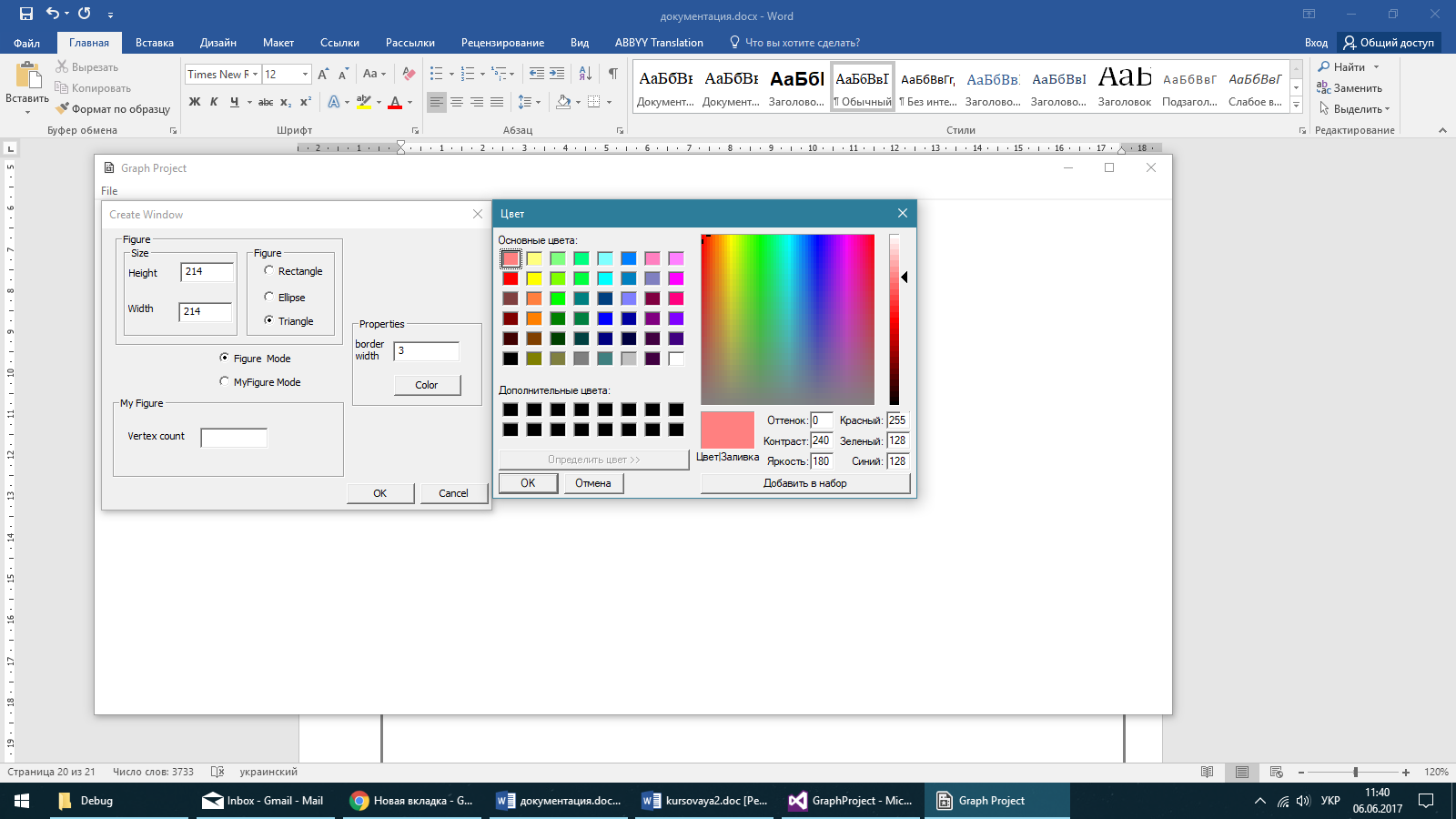
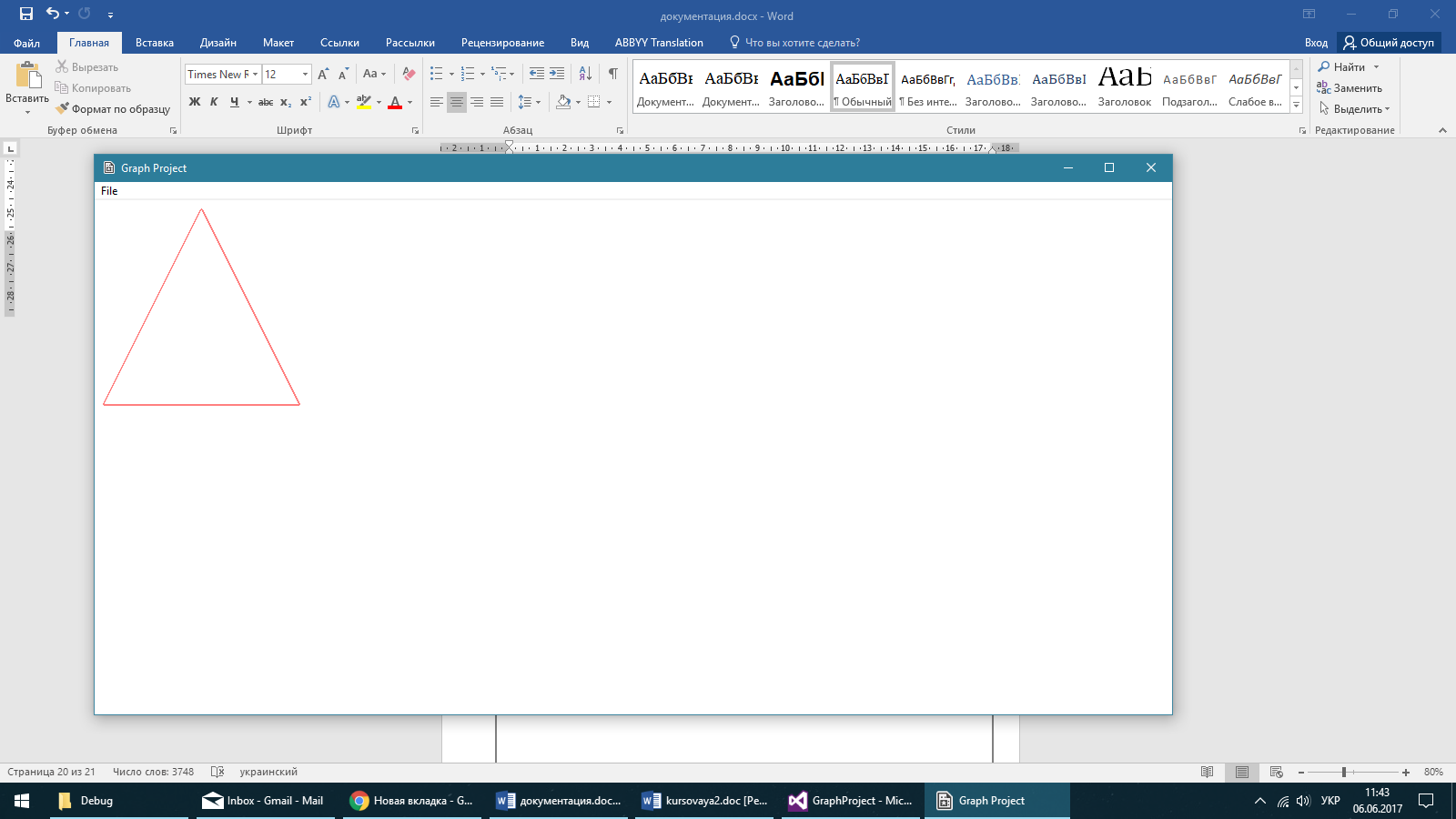
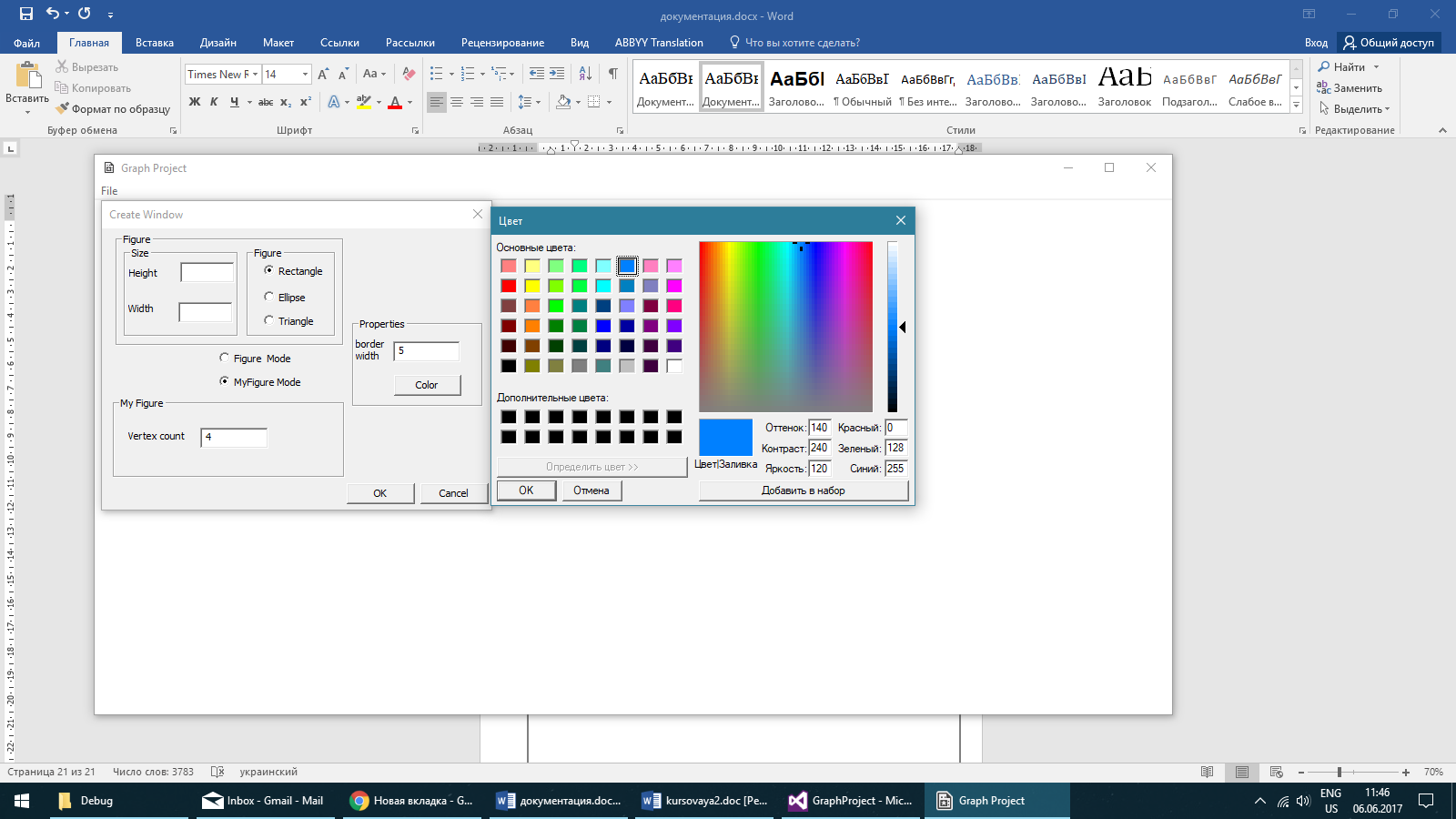
Для створення фігури яка передбачена програмою необхідно вибрати “Figure Mode” і, у груповому блоці Figure встановити відповідні значення, приклад на рисунку 2.10, і натиснути клавішу “Ok” в результаті фігура буде створена (див. рис. 2.11)

Рисунок 2.10 – Створення фігури

Рисунок 2.10 – Створена фігура

Для створення користувацької фігури, необхідно вибрати “MyFigure Mode” і, у груповому блоці Figure встановити відповідні значення, приклад на рисунку 2.12, і натиснути клавішу “Ok” в результаті фігура буде створена (див. рис. 2.13), і нарисована, коли користувач встановить координати вершин, натисканням ЛКМ на клієнтській області екрану.

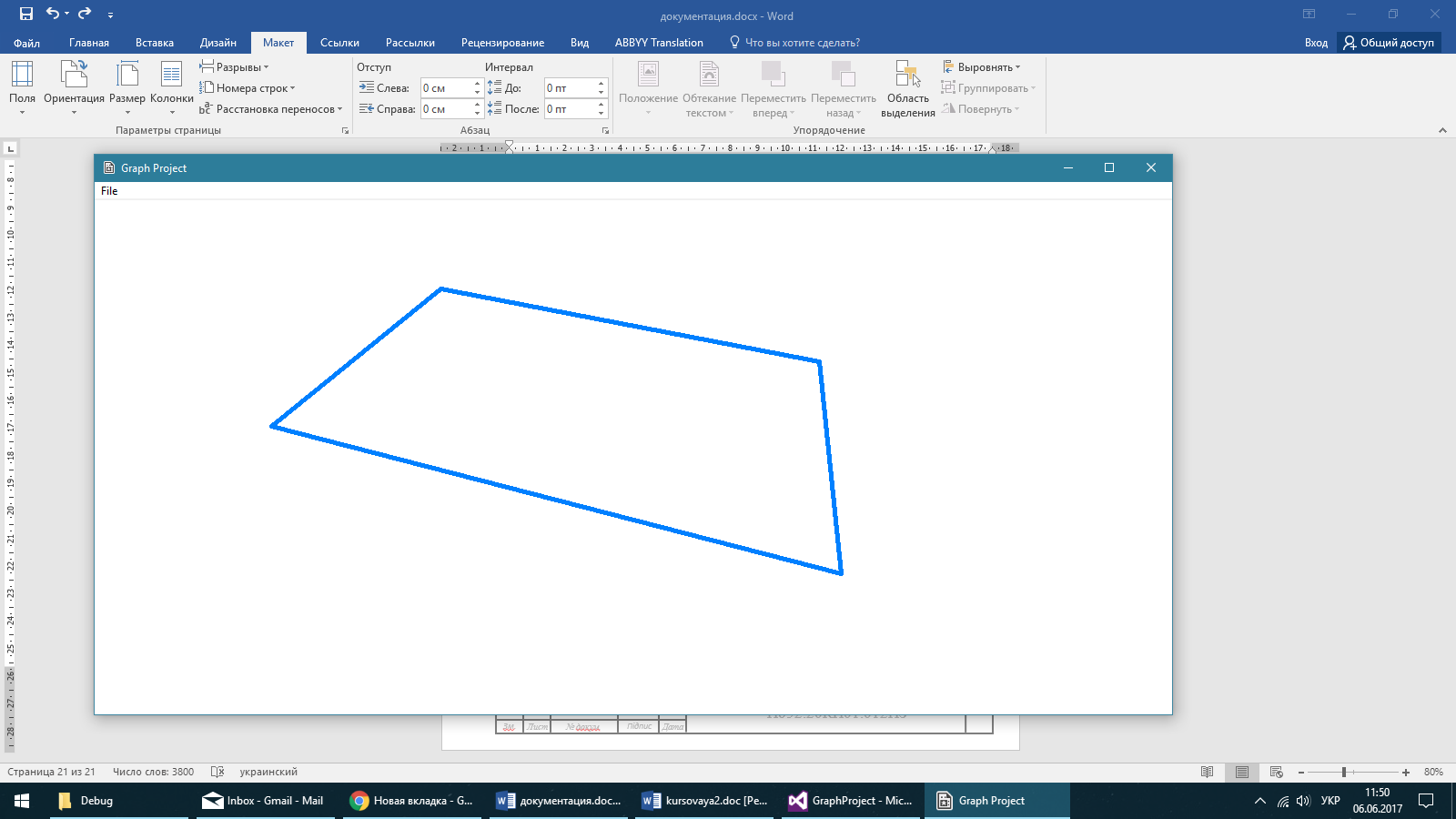
Рисунок 2.12 – Створення користувацької фігури

Рисунок 2.13 – Створена користувацька фігури

У випадку, коли користувач хоче перемикнути режим роботи програми, йому необхідно відкрити вікно створення фігур, та перемикнути перемикач у положення “MyFigure Mode ” або “Figure Mode” та натиснути кнопку “Ok” (рис. 2.9).

Якщо користувач хоче вибрати фігуру, йому необхідно натиснути на регіон екрану, який належить даній фігурі (див. рис 2.14.)

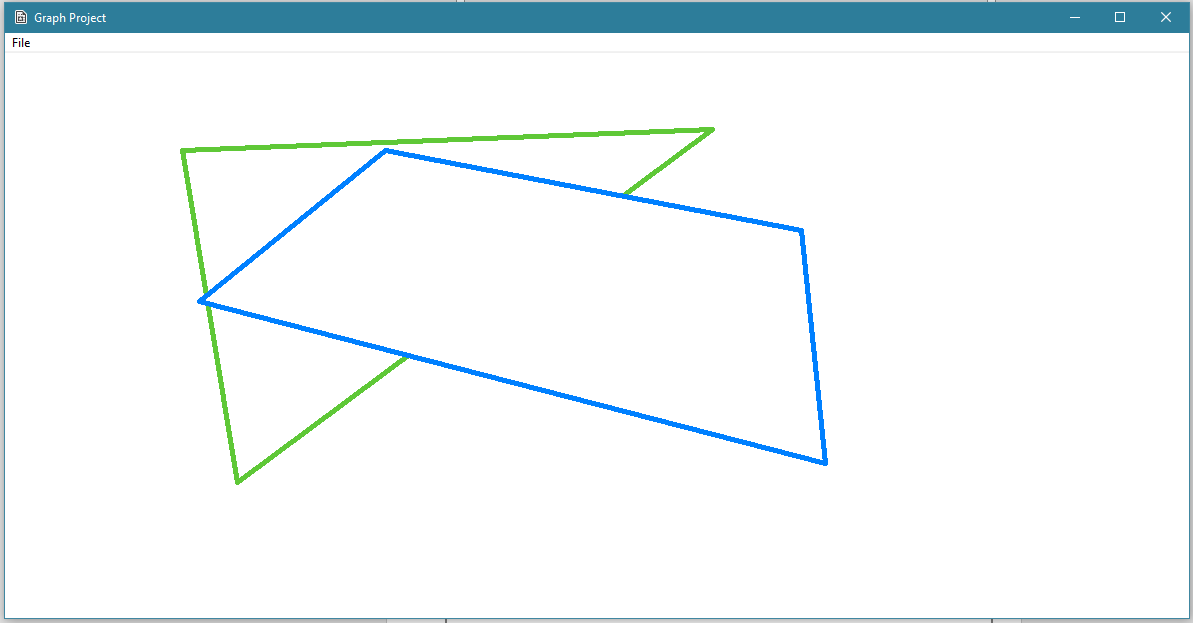
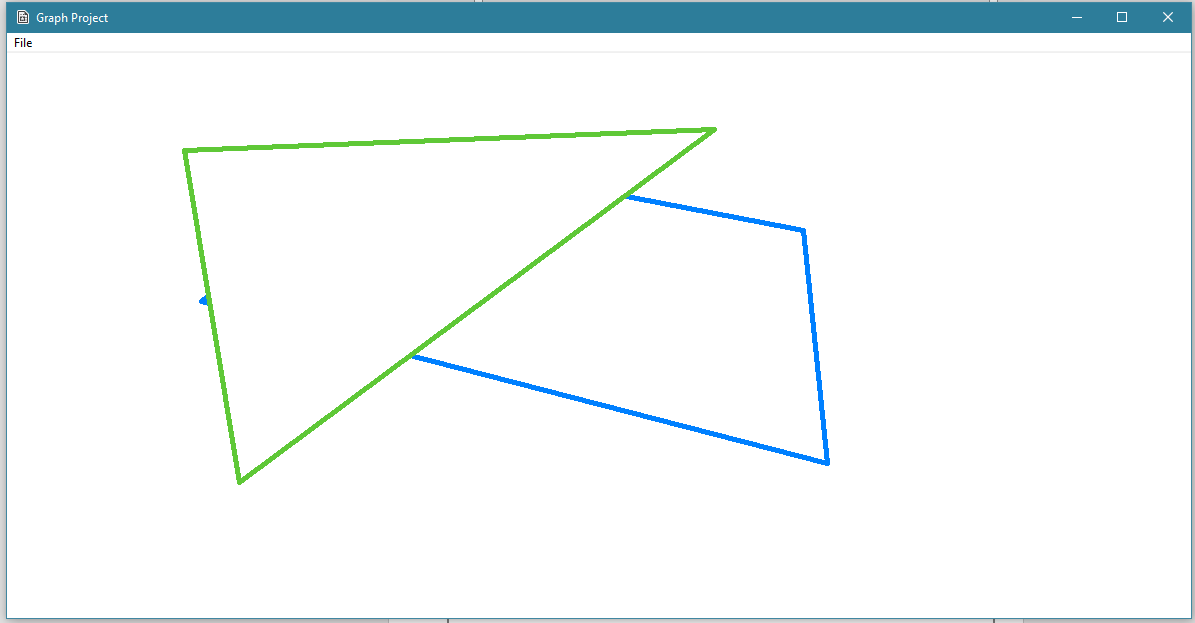
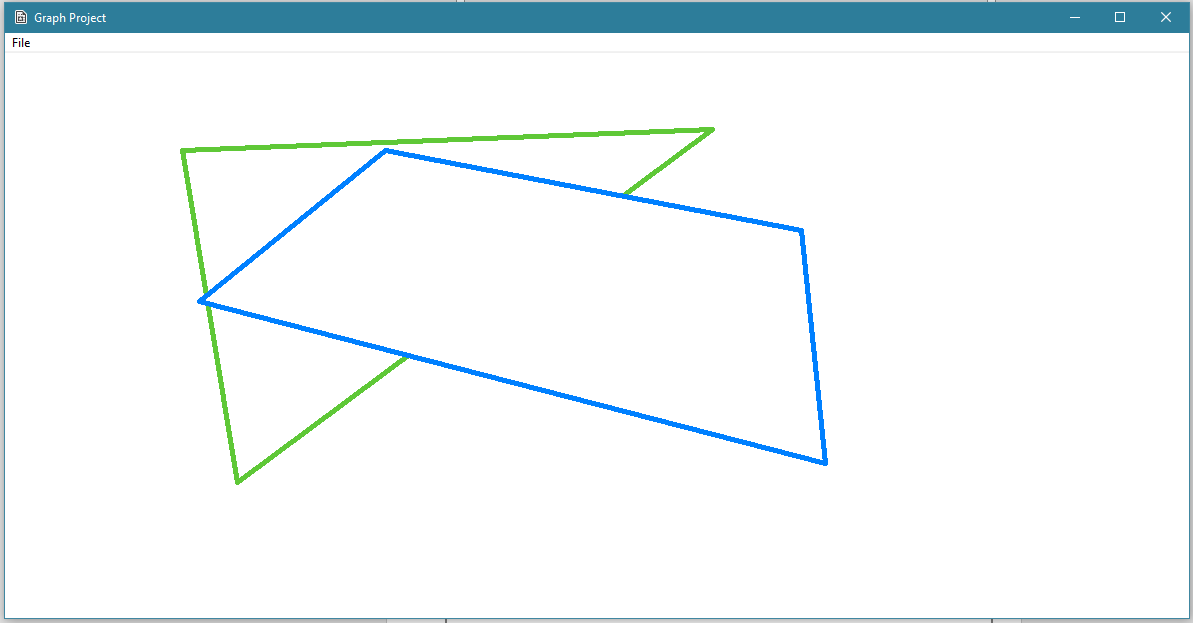
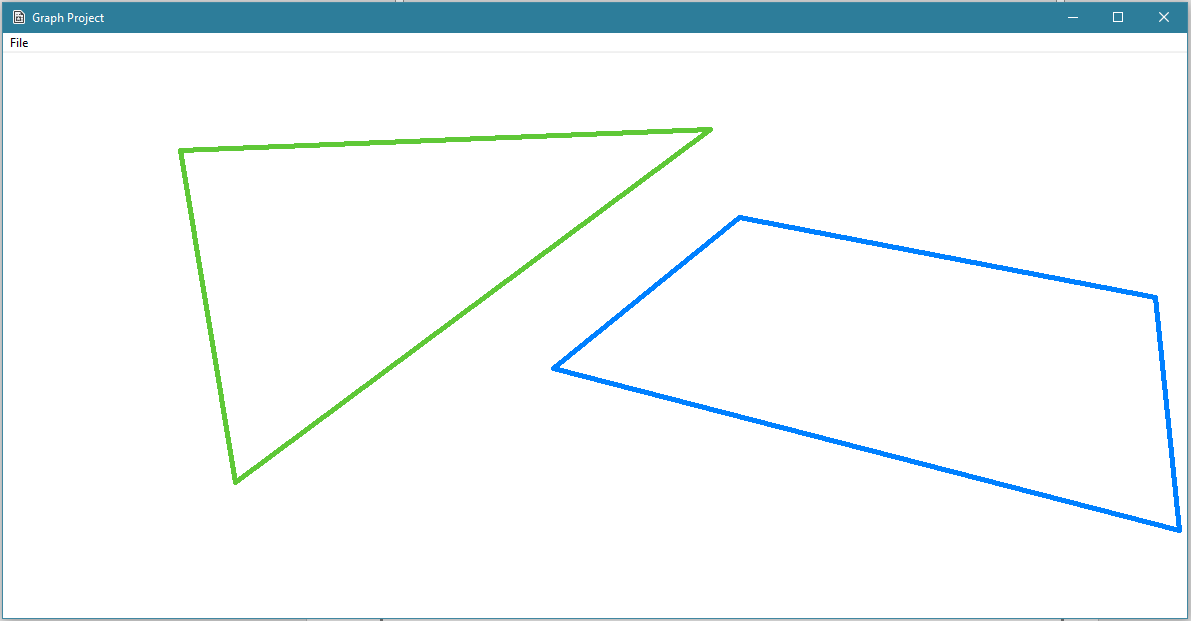


Рисунок 2.14 – Вибирання фігури

Щоб перетягнути фігуру, користувачу наобхідно вибрати її, і натиснувши ЛКМ на регіоні, що належить даній фігурі, тягнути миш (див. рис. 2.15).

Рисунок 2.15 – Зміна положення фігури



Видалення фігури користувач може виконати через меню (рис. 2.8). Для цього необхідно вибрати фігуру, а потім виконати меню-“File”-“Delete Figure” (див. рис. 2.16).

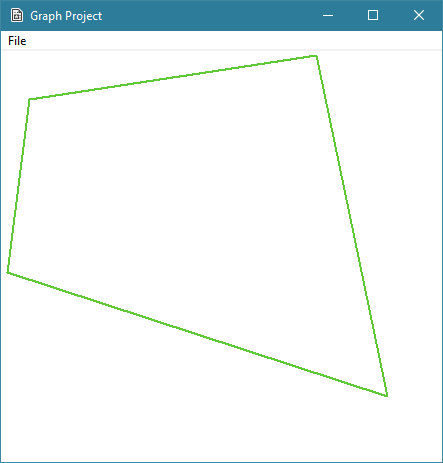
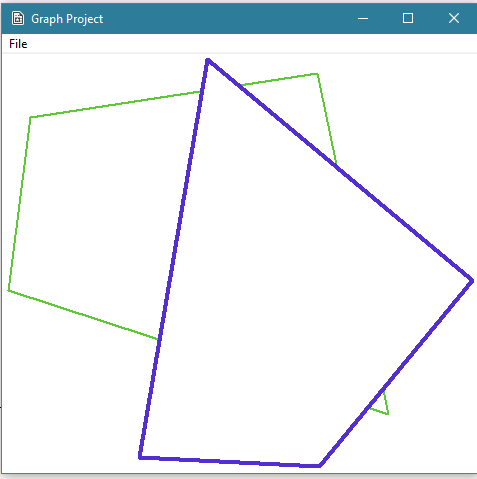


Рисунок 2.16 – Видалення фігури

**Висновок**

В ході написання курсової роботи була розроблена програма для Windows 10 на мові програмування Visual C++.

Ознайомився з різними видами графічних редакторів, та їх застосування в залежності від постановленої задачі. Сформував технічне завдання та план його виконання. Розробив готовий проект, який частково реалізує графічний редактор векторної графіки, і забезпечує певним функціоналом користувача.

Для написання цього проекту було обрано середовище програмування Visual Studio 2015, яке включає в себе всі необхідні елементи для реалізації поставленої задачі.

Під час виконання роботи я закріпив навички та уміння програмувати на Visual Studio з використанням Visual C++, а також отримав деякі нові навички у програмуванні.

**Список використаних джерел**

1. <https://uk.wikipedia.org/wiki/Графічний_редактор>
2. <https://uk.wikipedia.org/wiki/Windows_API>
3. <https://uk.wikipedia.org/wiki/Растровый_графический_редактор>
4. <https://uk.wikipedia.org/wiki/Векторный_графический_редактор>
5. <https://docs.altlinux.org>
6. Видавництво: БХВ-Петербург   
   Автор: Н.А. Литвиненко  
   Назва: Технологія програмування на С++ Win32 API – застосування  
   Кількість сторінок: 288  
   Рік видання: 2010
7. Видавництво: БХВ-Петербург   
   Автор: Н.А. Литвиненко  
   Назва: Технологія програмування на С++ початковий курс  
   Кількість сторінок: 280  
   Рік видання: 2005

**Додаток А**

**Код програми**

**// stdafx.h**

#pragma once

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <windowsx.h>

#include <stdlib.h>

#include <malloc.h>

#include <memory.h>

#include <tchar.h>

#include <commdlg.h>

#include <cmath>

**// main.h**

#pragma once

#include "WindowFunctions.h"

// main.cpp

#include "main.h"

HWND hWnd; // descriptor

HINSTANCE hInst;

WCHAR szTitleName[MAX\_LOADSTRING];

WCHAR szMainClassName[MAX\_LOADSTRING];

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitleName, MAX\_LOADSTRING);

LoadStringW(hInstance, IDS\_GRAPHPROJECT, szMainClassName, MAX\_LOADSTRING);

MyRegisterClass(hInstance, MainWndProc, szMainClassName);

// create window

CreateMainWnd(hWnd, hInst, nCmdShow, szMainClassName, szTitleName);

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDA\_ACCELERATOR));

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

**// WindowFunction.h**

#pragma once

#include "Rectangles.h"

#include "Ellipses.h"

#include "Triangles.h"

#include "Figure.h"

#include "MyFigure.h"

#include "ObjectManage.h"

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

ATOM MyRegisterClass(HINSTANCE hInstance, WNDPROC WndProc, WCHAR \*szClassName);

void CreateMainWnd(HWND & hWnd, HINSTANCE hInst, INT nCmdShow, LPWSTR ClassName, LPWSTR TitleName);

INT\_PTR CALLBACK DialogProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

**// WindowFunction.cpp**

#include "WindowFunctions.h"

INT nTopFigure, nTopMyFigure, nPtCounter; // top of windows stack

RECT WndRect; // window size

// Objects windows

Figure \*Window[MAX\_FIGURES];

MyFigure myWindow[MAX\_FIGURES];

/\* mode selector \*/

BOOL bModeFigure = true; BOOL bModeMyFigure = false;

/\*\*/

// dialog box

TCHAR szBuffEdtHeight[EDIT\_BUFFSIZE], szBuffEdtWidth[EDIT\_BUFFSIZE], // text buffers to dialogbox

szBuffEdtPenWidth[EDIT\_BUFFSIZE], szBuffEdtCount[EDIT\_BUFFSIZE];

INT nEdtHeight, nEdtWidth, nEdtPenWidth, nEdtCount; // value of edits

BOOL bRBRectChecked = false, bRBElpsChecked = false, bRBTrglChecked = false, bErase = false;

COLORREF PenColor = RGB(0, 0, 0);

CHOOSECOLOR colorDlg;

COLORREF acrCustClr[16];

// to draw

HPEN hWindowPen;

HDC hdc;

PAINTSTRUCT ps;

// difference between pointer and [x,y] position of figure

INT nCrntDiffX;

INT nCrntDiffY;

// it shows whether figure is captured or no

BOOL bCptred = false;

// screen buffer buffer

HBITMAP hBuff = NULL;

HDC hMem = NULL;

POINT ptMouse;// current mouse pt

POINT ptMousePrev;

POINT ptMouseOld; // old mouse point to return figure in case

ATOM MyRegisterClass(HINSTANCE hInstance, WNDPROC WndProc, WCHAR \*szClassName) {

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEXW);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc; // WNDPROC

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance; // HINSTANCE

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_GRAPHPROJECT\_ICON));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDM\_MENU);

wcex.lpszClassName = szClassName; // WCHAR

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_GRAPHPROJECT\_ICON));

return RegisterClassExW(&wcex);

}

void CreateMainWnd(HWND &hWnd, HINSTANCE hInst, INT nCmdShow, LPWSTR ClassName, LPWSTR TitleName)

{

hWnd = CreateWindowW(ClassName, TitleName, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInst, nullptr);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

}

void ColorDlgInit(HWND &hWnd) {

colorDlg.lStructSize = sizeof(CHOOSECOLOR);

colorDlg.hwndOwner = hWnd;

colorDlg.rgbResult = PenColor;

colorDlg.Flags = CC\_RGBINIT | CC\_FULLOPEN;

colorDlg.lpCustColors = (LPDWORD)acrCustClr;

}

INT\_PTR CALLBACK DialogProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

{

/\* Radio for figure init\*/

CheckRadioButton(hWnd, IDC\_RADIO\_RECTANGLE, IDC\_RADIO\_TRIANGLE, IDC\_RADIO\_RECTANGLE);

bRBRectChecked = true;

/\*\*/

/\* Radio for myFigure init\*/

if(bModeFigure)

CheckRadioButton(hWnd, IDC\_RADIO\_FIGUREMODE, IDC\_RADIO\_MYFIGUREMODE, IDC\_RADIO\_FIGUREMODE);

else

CheckRadioButton(hWnd, IDC\_RADIO\_FIGUREMODE, IDC\_RADIO\_MYFIGUREMODE, IDC\_RADIO\_MYFIGUREMODE);

bErase = false;

/\*\*/

/\* color dialog init\*/

ColorDlgInit(hWnd);

return (INT\_PTR)TRUE;

}

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDC\_RADIO\_RECTANGLE:

bRBRectChecked = true;

bRBElpsChecked = false;

bRBTrglChecked = false;

break;

case IDC\_RADIO\_ELLIPSE:

bRBRectChecked = false;

bRBElpsChecked = true;

bRBTrglChecked = false;

break;

case IDC\_RADIO\_TRIANGLE:

bRBRectChecked = false;

bRBElpsChecked = false;

bRBTrglChecked = true;

break;

case IDC\_RADIO\_FIGUREMODE:

if (bModeMyFigure) {

bModeMyFigure = false;

bErase = true;

}

bModeFigure = true;

break;

case IDC\_RADIO\_MYFIGUREMODE:

if (bModeFigure) {

bModeFigure = false;

bErase = true;

}

bModeMyFigure = true;

break;

case IDC\_BUTTON\_COLLOR:

if (ChooseColor(&colorDlg)) {

PenColor = colorDlg.rgbResult;

}

break;

case IDOK:

GetWindowText(GetDlgItem(hWnd, IDC\_EDIT\_HEIGHT), szBuffEdtHeight, EDIT\_BUFFSIZE);

GetWindowText(GetDlgItem(hWnd, IDC\_EDIT\_WIDTH), szBuffEdtWidth, EDIT\_BUFFSIZE);

GetWindowText(GetDlgItem(hWnd, IDC\_EDIT\_PENWIDTH), szBuffEdtPenWidth, EDIT\_BUFFSIZE);

GetWindowText(GetDlgItem(hWnd, IDC\_EDIT\_VERTEXCOUNT), szBuffEdtCount, EDIT\_BUFFSIZE);

case IDCANCEL:

EndDialog(hWnd, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

}

return (INT\_PTR)FALSE;

}

SIZE sizeDelta;

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

GetClientRect(hWnd, &WndRect);

nTopFigure = nTopMyFigure = -1; // no Window

hdc = GetDC(hWnd);

hMem = CreateCompatibleDC(hdc); // virtual dc, a buffer

hBuff = CreateCompatibleBitmap(hdc, WIDTH, HEIGHT); // bitmap

SelectObject(hMem, hBuff); // select Bitmap on virtual dc

PatBlt(hMem, 0, 0, WndRect.right, WndRect.bottom, WHITENESS);

ReleaseDC(hWnd, hdc);

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_FILE\_CREATE:

if (DialogBox(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD\_NEW), hWnd, DialogProc) == IDOK)

{

nEdtPenWidth = \_ttoi(szBuffEdtPenWidth);

if (bModeFigure)

{

nEdtHeight = \_ttoi(szBuffEdtHeight);

nEdtWidth = \_ttoi(szBuffEdtWidth);

if (bErase) {

if (nTopFigure > 0)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_ENABLED);

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

bErase = false;

}

if (nEdtHeight && nEdtWidth && nEdtPenWidth) {

if (!++nTopFigure)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_ENABLED);

if (bRBRectChecked)

{

Rectangles \*tmp = new Rectangles;

ObjectInit(tmp, nTopFigure);

bRBRectChecked = false;

}

if (bRBElpsChecked)

{

Ellipses \*tmp = new Ellipses;

ObjectInit(tmp, nTopFigure);

bRBElpsChecked = false;

}

if (bRBTrglChecked)

{

Triangles \*tmp = new Triangles;

ObjectInit(tmp, nTopFigure);

bRBTrglChecked = false;

}

}

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

}

if (bModeMyFigure)

{

if (bErase) {

if(nTopMyFigure > 0)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_ENABLED);

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

bErase = false;

}

if ((nEdtCount = \_ttoi(szBuffEdtCount)) && nEdtPenWidth) {

nPtCounter = 0;

if (!++nTopMyFigure)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_ENABLED);

myWindow[nTopMyFigure].setColor(PenColor);

myWindow[nTopMyFigure].setFrameWidth(nEdtPenWidth);

}

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

}

}

break;

case IDM\_FILE\_DELETE:

if (bModeFigure && nTopFigure > -1) {

delete Window[nTopFigure];

if(--nTopFigure == -1)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_GRAYED);

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

}

if (bModeMyFigure && nTopMyFigure > -1) {

myWindow[nTopMyFigure].init();//my Figure

if(--nTopMyFigure == -1)

EnableMenuItem(GetMenu(hWnd), IDM\_FILE\_DELETE, MF\_GRAYED);

MyObjectDraw(hMem, nTopFigure, hWindowPen, myWindow, hWnd, WndRect);

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_LBUTTONDOWN:

if ((nTopFigure > -1 && bModeFigure)

|| (nTopMyFigure > -1 && bModeMyFigure)) {

ptMousePrev.x = ptMouse.x = LOWORD(lParam);

ptMousePrev.y = ptMouse.y = HIWORD(lParam);

if (bModeFigure)

{

if (nTopFigure > 0)

{

for (int i = nTopFigure; i >= 0; i--) // changing place in stack

if (Window[i]->isClicked(ptMouse)) {

Figure \*tmp = Window[i];

for (int j = i; j < nTopFigure; j++)

{

Window[j] = Window[j + 1];

}

Window[nTopFigure] = tmp;

i = -1;

}

}

if (Window[nTopFigure]->isClicked(ptMouse))

{

ptMouseOld.x = Window[nTopFigure]->getX();

ptMouseOld.y = Window[nTopFigure]->getY();

nCrntDiffX = ptMouse.x - Window[nTopFigure]->getX();

nCrntDiffY = ptMouse.y - Window[nTopFigure]->getY();

bCptred = true;

SetCapture(hWnd);

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

}

}

if (bModeMyFigure)

{

if (nPtCounter < nEdtCount) {

if (nPtCounter == nEdtCount - 1) {

myWindow[nTopMyFigure].setCurrPt(ptMouse, nPtCounter);

myWindow[nTopMyFigure].setCurrPt(myWindow[nTopMyFigure].getCurrPt(0), ++nPtCounter);

myWindow[nTopMyFigure].initRgn(hMem);

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

}

else {

myWindow[nTopMyFigure].setCurrPt(ptMouse, nPtCounter);

nPtCounter++;

}

}

else {

ptMouseOld = ptMouse;

if (nTopMyFigure > 0)

{

for (int i = nTopMyFigure; i >= 0; i--) // changing place in stack

if (myWindow[i].isClicked(ptMouse)) {

MyFigure tmp = myWindow[i];

for (int j = i; j < nTopMyFigure; j++)

myWindow[j] = myWindow[j + 1];

myWindow[nTopMyFigure] = tmp;

i = -1;

}

}

if (myWindow[nTopMyFigure].isClicked(ptMouse)) { // перед сделать перестановку в стэке

sizeDelta.cx = myWindow[nTopMyFigure].getCurrPt(0).x - ptMousePrev.x;

sizeDelta.cy = myWindow[nTopMyFigure].getCurrPt(0).y - ptMousePrev.y;

myWindow[nTopMyFigure].setDelta(sizeDelta);

bCptred = true;

SetCapture(hWnd);

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

}

}

}

}

break;

case WM\_LBUTTONUP:

if (bCptred)

{

ptMouse.x = LOWORD(lParam);

ptMouse.y = HIWORD(lParam);

if (!PtInRect(&WndRect, ptMouse))

{

if (bModeFigure)

{

Window[nTopFigure]->setX(ptMouseOld.x);

Window[nTopFigure]->setY(ptMouseOld.y);

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

}

if (bModeMyFigure)

{

myWindow[nTopMyFigure].move(ptMouseOld, ptMousePrev, hMem);

myWindow[nTopMyFigure].initRgn(hMem);

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

}

}

ReleaseCapture();

bCptred = false;

}

break;

case WM\_MOUSEMOVE:

if (bCptred)

{

ptMouse.x = LOWORD(lParam);

ptMouse.y = HIWORD(lParam);

if (bModeFigure)

{

int tmpX = ptMouse.x - nCrntDiffX,

tmpY = ptMouse.y - nCrntDiffY;

Window[nTopFigure]->setX(tmpX);

Window[nTopFigure]->setY(tmpY);

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

}

if (bModeMyFigure)

{

myWindow[nTopMyFigure].move(ptMouse, ptMousePrev, hMem);

myWindow[nTopMyFigure].initRgn(hMem);

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

ptMousePrev = ptMouse;

}

}

break;

case WM\_SIZE:

GetClientRect(hWnd, &WndRect);

if (bModeFigure)

ObjectDraw(hMem, nTopFigure, hWindowPen, Window, hWnd, WndRect);

else

MyObjectDraw(hMem, nTopMyFigure, hWindowPen, myWindow, hWnd, WndRect);

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

BitBlt(hdc, 0, 0, WndRect.right, WndRect.bottom, hMem, 0, 0, SRCCOPY);

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

DeleteObject(hMem);

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

break;

}

return 0;

}

**// ObjectManage.h**

#pragma once

#include "MyFigure.h"

#include "Figure.h"

void ObjectDraw(HDC &hMem, INT nTop, HPEN &hPen, Figure \*\*cWindow, HWND &hWnd, RECT &WndRect);

void MyObjectDraw(HDC &hMem, INT nTop, HPEN &hPen, MyFigure \*cWindow, HWND &hWnd, RECT &WndRect);

template <class TypeClass>

void ObjectInit(TypeClass \*tmp, int &nTopFigure)

{

tmp->setFrameWidth(nEdtPenWidth);

tmp->setColor(PenColor);

tmp->setX(10);

tmp->setY(10);

tmp->setHeight(nEdtHeight);

tmp->setWidth(nEdtWidth);

tmp->setFrameWidth(nEdtPenWidth);

Window[nTopFigure] = tmp;

}

// ObjectManage.cpp

#include "ObjectManage.h"

void ObjectDraw(HDC &hMem, INT nTop, HPEN &hPen, Figure \*\*cWindow, HWND &hWnd, RECT &WndRect) {

PatBlt(hMem, 0, 0, WndRect.right, WndRect.bottom, WHITENESS);

for (int i = 0; i <= nTop; i++)

cWindow[i]->draw(hMem, hPen);

InvalidateRect(hWnd, NULL, FALSE);

}

void MyObjectDraw(HDC & hMem, INT nTop, HPEN & hPen, MyFigure \* cWindow, HWND & hWnd, RECT & WndRect)

{

PatBlt(hMem, 0, 0, WndRect.right, WndRect.bottom, WHITENESS);

for (int i = 0; i <= nTop; i++)

cWindow[i].draw(hMem, hPen);

cWindow[nTop].initRgn(hMem);

InvalidateRect(hWnd, NULL, FALSE);

}

**// Figure.h**

#pragma once

#include "stdafx.h"

#include "resource.h"

class MyFigure

{

private:

POINT pt[MAX\_POINTS];

SIZE ptDelta[MAX\_POINTS-1];

HRGN hMyRegion;

COLORREF color;

int nFrameWidth;

int nTopPt;

public:

MyFigure();

void init();

void setCurrPt(POINT in\_pt, int i);

POINT getCurrPt(int i);

void draw(HDC &hdc, HPEN &hPen);

void move(POINT & curPt, POINT & prevPt, HDC &hdc);

void initRgn(HDC &hdc);

bool isClicked(POINT &pt);

void setColor(COLORREF color);

void setFrameWidth(int width);

void setDelta(SIZE delta);

~MyFigure();

};

**// Figure.cpp**

#include "Figure.h"

Figure::Figure()

{

}

void Figure::setColor(COLORREF color)

{

this->color = color;

}

COLORREF Figure::getColor()

{

return COLORREF(this->color);

}

void Figure::setFrameWidth(int width)

{

this->nFrameWidth = width;

}

int Figure::getFrameWidth()

{

return this->nFrameWidth;

}

void Figure::setHeight(int height)

{

this->height = height;

}

void Figure::setWidth(int width)

{

this->width = width;

}

void Figure::setX(int x)

{

this->pos.left = x;

}

void Figure::setY(int y)

{

this->pos.top = y;

}

int Figure::getX()

{

return pos.left;

}

int Figure::getY()

{

return pos.top;

}

Figure::~Figure()

{

}

**// Rectangle.h**

#pragma once

#include "Figure.h"

class Rectangles

: public Figure

{

public:

Rectangles();

void draw(HDC & hdc, HPEN &hPen);

bool isClicked(POINT &pt);

~Rectangles();

};

**// Rectangle.cpp**

#include "Rectangles.h"

Rectangles::Rectangles()

{

}

void Rectangles::draw(HDC &hdc, HPEN &hPen)

{

if (hPen) DeleteObject(hPen);

hPen = CreatePen(PS\_SOLID, nFrameWidth, color);

SelectObject(hdc, hPen);

this->pos.right = pos.left + width;

this->pos.bottom = pos.top + height;

Rectangle(hdc, pos.left, pos.top, pos.right, pos.bottom);

}

bool Rectangles::isClicked(POINT & pt)

{

return PtInRect(&pos, pt);

}

Rectangles::~Rectangles()

{

}

**// Triangle.h**

#pragma once

#include "Figure.h"

class Triangles

: public Figure

{

private:

HRGN hTriangleRgn;

POINT ptTriangle[3];

void initRgn(HDC &hdc);

public:

Triangles();

void draw(HDC & hdc, HPEN &hPen);

bool isClicked(POINT &pt);

~Triangles();

};

**// Triangle.cpp**

#include "Triangles.h"

Triangles::Triangles()

{

}

void Triangles::initRgn(HDC &hdc)

{

ptTriangle[0].x = pos.left;

ptTriangle[0].y = pos.bottom = pos.top + height;

ptTriangle[1].x = pos.left + (width / 2);

ptTriangle[1].y = pos.top;

ptTriangle[2].x = pos.right = pos.left + width;

ptTriangle[2].y = pos.bottom;

BeginPath(hdc);

Polyline(hdc, ptTriangle, 3);

CloseFigure(hdc);

EndPath(hdc);

if (hTriangleRgn) DeleteObject(hTriangleRgn);

hTriangleRgn = PathToRegion(hdc);

}

void Triangles::draw(HDC & hdc, HPEN & hPen)

{

if (hPen) DeleteObject(hPen);

hPen = CreatePen(PS\_SOLID, nFrameWidth, color);

SelectObject(hdc, hPen);

initRgn(hdc);

MoveToEx(hdc, ptTriangle[0].x, ptTriangle[0].y, NULL);

LineTo(hdc, ptTriangle[1].x, ptTriangle[1].y);

LineTo(hdc, ptTriangle[2].x, ptTriangle[2].y);

LineTo(hdc, ptTriangle[0].x, ptTriangle[0].y);

PaintRgn(hdc, hTriangleRgn);

}

bool Triangles::isClicked(POINT & pt)

{

return PtInRegion(hTriangleRgn, pt.x, pt.y);

}

Triangles::~Triangles()

{

}

**// Ellipse.h**

#pragma once

#include "Figure.h"

class Ellipses

: public Figure

{

private:

HRGN hEllipseRgn;

void initRgn();

public:

Ellipses();

void draw(HDC & hdc, HPEN &hPen);

bool isClicked(POINT &pt);

~Ellipses();

};

**// Ellipse.cpp**

#include "Ellipses.h"

void Ellipses::initRgn()

{

if (hEllipseRgn) DeleteObject(hEllipseRgn);

hEllipseRgn = CreateEllipticRgnIndirect(&this->pos);

}

Ellipses::Ellipses()

{

}

void Ellipses::draw(HDC & hdc, HPEN & hPen)

{

if (hPen) DeleteObject(hPen);

hPen = CreatePen(PS\_SOLID, nFrameWidth, color);

SelectObject(hdc, hPen);

this->pos.right = pos.left + width;

this->pos.bottom = pos.top + height;

Ellipse(hdc, pos.left, pos.top, pos.right, pos.bottom);

}

bool Ellipses::isClicked(POINT &pt)

{

this->initRgn();

return PtInRegion(hEllipseRgn, pt.x, pt.y);

}

Ellipses::~Ellipses()

{

if (hEllipseRgn) DeleteObject(hEllipseRgn);

}

**// MyFigure.h**

#pragma once

#include "stdafx.h"

#include "resource.h"

class MyFigure

{

private:

POINT pt[MAX\_POINTS];

SIZE ptDelta[MAX\_POINTS-1];

HRGN hMyRegion;

COLORREF color;

int nFrameWidth;

int nTopPt;

public:

MyFigure();

void init();

void setCurrPt(POINT in\_pt, int i);

POINT getCurrPt(int i);

void draw(HDC &hdc, HPEN &hPen);

void move(POINT & curPt, POINT & prevPt, HDC &hdc);

void initRgn(HDC &hdc);

bool isClicked(POINT &pt);

void setColor(COLORREF color);

void setFrameWidth(int width);

void setDelta(SIZE delta);

~MyFigure();

};

**// MyFigure.cpp**

#include "MyFigure.h"

MyFigure::MyFigure()

{

}

void MyFigure::init()

{

for (int i = 0; i < nTopPt; i++) {

pt[i] = { 0 };

ptDelta[i] = { 0 };

}

if (hMyRegion) DeleteObject(hMyRegion);

color = 0;

nFrameWidth = 0;

nTopPt = 0;

}

void MyFigure::setCurrPt(POINT in\_pt, int i)

{

pt[i] = in\_pt;

if(i > 0) {

ptDelta[i].cx = pt[0].x - in\_pt.x;

ptDelta[i].cy = pt[0].y - in\_pt.y;

}

nTopPt++;

}

POINT MyFigure::getCurrPt(int i)

{

return POINT(pt[i]);

}

void MyFigure::initRgn(HDC &hdc)

{

/\* creating path \*/

BeginPath(hdc);

Polyline(hdc, pt, nTopPt);

CloseFigure(hdc);

EndPath(hdc);

/\*\*/

/\* convert to Region\*/

if (hMyRegion) DeleteObject(hMyRegion);

hMyRegion = PathToRegion(hdc);

/\*\*/

}

void MyFigure::draw(HDC & hdc, HPEN & hPen)

{

if (hPen) DeleteObject(hPen);

hPen = CreatePen(PS\_SOLID, nFrameWidth, color);

SelectObject(hdc, hPen);

PaintRgn(hdc, hMyRegion);

Polyline(hdc, pt, nTopPt);

}

void MyFigure::move(POINT &curPt, POINT & prevPt, HDC &hdc)

{

pt[0].x = curPt.x + ptDelta[0].cx;

pt[0].y = curPt.y + ptDelta[0].cy;

for (int i = 1; i < nTopPt; i++)

{

pt[i].x = pt[0].x - ptDelta[i].cx;

pt[i].y = pt[0].y - ptDelta[i].cy;

}

}

bool MyFigure::isClicked(POINT &pt)

{

return PtInRegion(hMyRegion, pt.x, pt.y);

}

void MyFigure::setColor(COLORREF color)

{

this->color = color;

}

void MyFigure::setFrameWidth(int width)

{

this->nFrameWidth = width;

}

void MyFigure::setDelta(SIZE delta)

{

this->ptDelta[0] = delta;

}

MyFigure::~MyFigure()

{

}