

Міністерство освіти і науки України  
Полтавський національний педагогічний університет імені В.Г. Короленка

**Кривцова Олена Павлівна**

**Програмування мовою С++.  
Технологія візуального програмування**

навчальний посібник

м. Полтава – 2020

**УДК 004.43(075.8)**

**K82**

Рекомендований Вченою радою Полтавського національного педагогічного університету імені В.Г. Короленка (протокол №\_\_\_\_\_ від \_\_\_\_\_ 2020р.)

**Рецензенти:**

**Ємець О.О.** - доктор фіз.-мат. наук, завідувач кафедри математичного моделювання та соціальної інформатики Полтавського університету економіки і торгівлі

**Барболіна Т.М.** - доктор фіз.-мат. наук, зав.кафедри математичного аналізу та інформатики Полтавського національного педагогічного університету імені В.Г. Короленка

**Кривцова О.П.**

K82 Програмування мовою C++. Технологія візуального програмування : навч. посіб. – Полтава : ПНПУ імені В.Г. Короленка, 2020. – 144 с.

**УДК 004.43(075.8)**

Навчальний посібник «Програмування мовою C++. Технологія візуального програмування» висвітлює основні можливості реалізації технології візуального програмування засобами мови C++. Розглянуто ряд тем курсу «Програмування» та «Інформатика» із прикладами розробки Windows-додатків у середовищі Microsoft Visual C++ Express.

На простих прикладах показано особливості створення Windows-додатків. Всі приклади супроводжуються детальними коментарями. До кожної теми запропоновано задачі для самостійної роботи.

© Кривцова О.П, 2020

© Полтава, 2020

# ЗМІСТ

<b>ВСТУП.....</b>	<b>5</b>
<b>ТЕМА 1. ВІЗУАЛЬНЕ СЕРЕДОВИЩЕ РОЗРОБКИ ПРОГРАМ МОВОЮ C++.</b>	<b>7</b>
ТЕХНОЛОГІЯ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ .....	7
СЕРЕДОВИЩЕ MICROSOFT VISUAL C++ EXPRESS .....	9
ЕТАПИ РОЗРОБКИ ДОДАТКІВ .....	13
МОЖЛИВОСТІ ВІДЛАГОДЖЕННЯ WINDOWS-ДОДАТКІВ .....	16
<b>ТЕМА 2. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ.....</b>	<b>20</b>
ОСОБЛИВОСТІ РОБОТИ З ФОРМАМИ.....	20
ХАРАКТЕРИСТИКА ОСНОВНИХ КОМПОНЕНТІВ СЕРЕДОВИЩА .....	21
КОМПОНЕНТ BUTTON.....	23
КОМПОНЕНТ LABEL.....	25
КОМПОНЕНТ TEXTBOX .....	26
КОМПОНЕНТИ CHECKBOX І RADIOBUTTON.....	29
<b>ТЕМА 3. РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА .....</b>	<b>31</b>
КОМПОНЕНТ MENUSTRIP .....	31
КОМПОНЕНТ CONTEXTMENUSTRIP.....	33
КОМПОНЕНТ TOOLSTRIP .....	34
КОМПОНЕНТ STATUSSTRIP.....	35
СТАНДАРТНІ ДІАЛОГОВІ ВІКНА.....	36
<i>Компонент OpenFileDialog.....</i>	<i>36</i>
<i>Компонент SaveFileDialog.....</i>	<i>37</i>
<b>ТЕМА 4. ГРАФІЧНІ МОЖЛИВОСТІ МОВИ C++ .....</b>	<b>39</b>
КОМПОНЕНТ PICTUREBOX .....	39
ВИВЕДЕННЯ ГРАФІЧНИХ ЗОБРАЖЕНЬ.....	41
ОЛІВЦІ І ПЕНЗЛІ .....	41
ГРАФІЧНІ ПРИМІТИВИ .....	45
<i>Методи побудови графічних примітивів .....</i>	<i>47</i>
<i>Лінія.....</i>	<i>48</i>
<i>Ламана лінія .....</i>	<i>48</i>
<i>Прямокутник.....</i>	<i>49</i>
<i>Точка .....</i>	<i>50</i>
<i>Багатокутник.....</i>	<i>50</i>
<i>Еліпс і коло .....</i>	<i>51</i>
<i>Дуга .....</i>	<i>53</i>
<i>Сектор.....</i>	<i>53</i>

Текст.....	54
МЕТОД БАЗОВОЇ ТОЧКИ .....	55
АНІМАЦІЯ .....	56
ЕТАПИ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ .....	58
<b>ЛАБОРАТОРНІ РОБОТИ .....</b>	<b>70</b>
ЛАБОРАТОРНА РОБОТА №1. ОЗНАЙОМЛЕННЯ З ВІЗУАЛЬНИМ СЕРЕДОВИЩЕМ VISUAL C++.	
КОМПІЛЯЦІЯ ТА ЗАПУСК ПРОГРАМИ НА ВИКОНАННЯ. ....	70
ЛАБОРАТОРНА РОБОТА №2. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ. ....	76
ЛАБОРАТОРНА РОБОТА №3 НАЛАШТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА.....	80
ЛАБОРАТОРНА РОБОТА №4 РОБОТА ІЗ ЗМІННИМИ ЧИСЛОВИХ ТА РЯДКОВИХ ТИПІВ. ....	83
ЛАБОРАТОРНА РОБОТА №5 РОЗРОБКА ДОДАТКІВ З ВИКОРИСТАННЯМ ЕЛЕМЕНТІВ УПРАВЛІННЯ. ....	88
ЛАБОРАТОРНА РОБОТА №6 СТВОРЕННЯ ГОЛОВНОГО ТА КОНТЕКСТНОМУ МЕНЮ ДОДАТКІВ.....	91
ЛАБОРАТОРНА РОБОТА №7 КОМПОНЕНТИ ЗОВНІШНЬОГО ОФОРМЛЕННЯ. ПАНЕЛІ ІНСТРУМЕНТІВ .....	96
ЛАБОРАТОРНА РОБОТА №8. РОЗРОБКА ДІАЛОГОВИХ ВІКОН. СИСТЕМНІ ДІАЛОГИ .....	98
ЛАБОРАТОРНА РОБОТА №9 ОСОБЛИВОСТІ РОБОТИ З ГРАФІЧНИМИ ПРИМІТИВАМИ .....	103
ЛАБОРАТОРНА РОБОТА №10 МЕТОД БАЗОВОЇ ТОЧКИ .....	106
ЛАБОРАТОРНА РОБОТА №11 СТВОРЕННЯ АНІМАЦІЇ ЗАСОБАМИ МОВИ C++.....	110
<b>ПРАКТИЧНІ РОБОТИ .....</b>	<b>114</b>
ПРАКТИЧНА РОБОТА №1 ОРГАНІЗАЦІЯ ПРОЕКТУ. ВІДЛАГОДЖЕННЯ ДОДАТКУ.....	114
ПРАКТИЧНА РОБОТА №2 ОСОБЛИВОСТІ ОБРОБКИ ТАБЛИЧНИХ ДАНИХ .....	117
ПРАКТИЧНА РОБОТА №3 ОСОБЛИВОСТІ РОЗРОБКИ ПРИКЛАДНИХ ДОДАТКІВ .....	120
ПРАКТИЧНА РОБОТА №4 ІНІЦІУВАННЯ ОБРОБКИ ПОДІЇ КЛАВІАТУРИ .....	127
ПРАКТИЧНА РОБОТА №5 ОСОБЛИВОСТІ РОБОТИ З ГРАФІЧНИМИ ДАНИМИ .....	128
<b>ДОДАТКИ .....</b>	<b>136</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>	<b>144</b>

## ВСТУП

Із стрімким розвитком апаратного забезпечення різко збільшується кількість програмних засобів, які використовують як початківці, так і їх розробники. Глобальна інформатизація суспільства зумовлює збільшення обсягів існуючої інформації і потужності комп'ютерів та комп'ютерних мереж для роботи з нею.

Будь яку комп'ютерну програму можна розглядати, як запис алгоритму розв'язання задачі у вигляді послідовності команд або операторів мовою, яку розуміє комп'ютер.

Для нормального розв'язання задач на комп'ютері потрібно, щоб програма була налагоджена, не потребувала доопрацювань і мала відповідну документацію. Тому стосовно роботи на комп'ютер часто використовують термін «програмний засіб». Даний термін можна визначити, як програму або сукупність програм на носіїві даних із програмною документацією, розроблених відповідно до стандартів й інших нормативних документів і придатних для використання за своїм призначенням [3].

Сукупність програм, процедур і правил, а також документації, що стосуються функціонування системи обробки даних називають програмним забезпеченням. Програмне забезпечення ПК поділяють на такі основні класи:

1. Базове ПЗ (операційна система та сервісні програми).
2. Інструментальні мови і системи програмування.
3. Прикладне програмне забезпечення.

Прикладні системи або **прикладне програмне забезпечення (ППЗ)** - це системи призначені для розв'язання задачі чи класу задач, або для надання користувачеві певних послуг. Завдяки прикладним системам можуть розв'язувати свої професійні задачі навіть користувачі комп'ютерів, які не вміють програмувати.

Аналізуючи основні вимоги до розробки прикладних додатків найважливішим чинником є кінцевий користувач. Тому суттєвими ми вважаємо такі вимоги до ППЗ:

1. Максимум зручностей користувача, що передбачає:
  - спілкування мовою, близькою до природної;
  - наочне представлення даних, можливість їх редагування;
  - швидкість ознайомлення з роботою, легкість освоєння;
  - відсутність жорстких обмежень на структуру та обсяг вихідних даних;

- можливість адаптації до вимог користувача;
- повнота і доступність програмної документації;

2. Адаптованість ПЗ - пристосованість до функціонування в різних умовах.

3. Гнучкість - можливість легко вводити зміни, доповнення та виправлення до ПЗ.

4. Мобільність - переносимість на різні обчислювальні платформи та операційні середовища.

5. Ефективність роботи.

Серед основних принципів побудови ПЗ слід відзначити [8]:

1. Модульність ПЗ - проведення декомпозиції алгоритмів і програм на модулі з метою виділення загальних типових функцій і компонентів.

2. Інтелектуальність ПЗ - наявність знань про предметну область і вміння використовувати їх при вирішенні завдань.

3. Чорний ящик - ПЗ повинно приховувати від користувачів складний механізм організації та взаємодії програм.

Отже для того, щоб реалізувати прикладний додаток високого рівня, необхідно забезпечити розробку якісного, ефективного та стабільного програмного продукту. Зазвичай стабільність досягається шляхом використання прийомів модульного програмування, зокрема, виділення модулів та визначення інтерфейсів.

# ТЕМА 1. ВІЗУАЛЬНЕ СЕРЕДОВИЩЕ РОЗРОБКИ ПРОГРАМ МОВОЮ C++

## Технологія візуального програмування

Термін технологія походить від грецьких техно - мистецтво, ремесло, наука і лого - поняття, навчання.

Технологія програмування являє собою сукупність узагальнених і систематизованих знань, або наука про оптимальні способи програмування, що забезпечує одержання програмної продукції з заданими властивостями.

Історичному контексті, виділяючи основні етапи розвитку програмування як науки [14]:

**Перший етап – “стихийне” програмування.** Цей етап охоплює період від моменту появи перших обчислювальних машин до середини 60-их років ХХ ст.

**Другий етап – структурний підхід до програмування (60 – 70 роки ХХ ст.).** *Структурний підхід до програмування* представляє собою сукупність технологічних прийомів, що охоплюють виконання всіх етапів розробки програмного забезпечення. В основі структурного підходу лежить *декомпозиція* (розбиття на частини) складних систем з ціллю наступної реалізації у вигляді окремих невеликих підпрограм. З появою інших принципів декомпозиції даний спосіб отримав назву *процедурної декомпозиції*.

**Третій етап – об’єктний підхід до програмування (з середини 80-х до кінця 90-х років ХХ ст.).** *Об’єктно-орієнтоване програмування* визначається як технологія створення складного програмного забезпечення, основана на представленні програми у вигляді сукупності об’єктів, кожен із яких є екземпляром визначеного типу (*класу*), а класи утворюють ієрархію з унаслідуванням властивостей. Взаємодія програмних об’єктів в такій системі відбувається шляхом передачі *повідомлень*.

Бурний розвиток технологій програмування, в основі яких лежить об’єктний підхід, дозволив розв’язати багато проблем. Так, були створені середовища, які підтримують *візуальне програмування*, наприклад Delphi, C++ Builder, Visual C++ і

т.д. При використанні візуального середовища у програміста з'являється можливість проектувати деяку частину, наприклад інтерфейси майбутнього продукту, з використанням візуальних засобів додавання і налаштування спеціальних бібліотечних компонентів. Результатом візуального проектування є заготовка майбутньої програми, в яку вже внесені відповідні коди.

**Четвертий етап – компонентний підхід і CASE-технології (з середини 90-х років ХХ ст. до нашого часу).** *Компонентний підхід* пропонує побудову програмного забезпечення з окремих компонентів фізично окремо існуючих частин програмного забезпечення, які взаємодіють між собою через *стандартизовані двійкові інтерфейси*. На відміну від звичайних об'єктів об'єкти-компоненти можна збирати в динамічні бібліотеки або файли які виконуються, розповсюджувати в двійковому коді і використовувати в будь-якій мові програмування, що підтримує відповідну технологію.

Перехід до технології візуального програмування вимагав створення нової бібліотеки класів. Головною особливістю даної бібліотеки полягає в тому, що в її основі лежить концепція **властивостей, методів та подій**.

**Властивості** є атрибутами компоненту, що визначають його зовнішній вигляд і поведінку. Багато властивостей компоненту мають значення, що встановлюється за замовчуванням (*наприклад розмір кнопки*). Можна визначити властивості під час проектування або написати код для видозміни властивостей компоненту під час виконання додатку (*видимий чи не видимий компонент в залежності від виконання умови*).

Властивості об'єкту визначають його характеристики та поведінку.

**Ім'я об'єкту -> властивість = значення властивості;**

*Наприклад: **label1->Text = "текстове повідомлення"***

Другим елементом для управління об'єктами є **методи**. На відміну від властивостей, які представляють собою змінні чи об'єкти, методи представляють собою функції класу (об'єкту), які можна викликати з зовні для управління об'єктом.

**Ім'я об'єкту -> Метод();**



Метод це функцією, яка пов'язана з компонентом, і яка оголошується як частина об'єкту.

*Наприклад: **button1-> Hide()** //приховати кнопку.*

Повний перелік властивостей і методів об'єкту приводиться в документації та у в будованій довідковій системі.

Третій елемент управління об'єктом – це **подія**. Сама операційна система Windows – це система яка управляється подіями. Для кожного об'єкту існує свій перелік подій, для яких можна змінити стандартну реакцію програми. Змінити реакцію на обробку події можна використовуючи вкладку подій у вікні властивостей.

Створюючи обробник події, ви доручаєте програмі виконати написану функцію, якщо ця подія відбудеться.

## **Середовище Microsoft Visual C++ Express**

Після завантаження на екрані з'являється інтегроване середовище розробки.

До основних елементів середовища програмування відносяться (*рис.1*):

**Команда головного меню (1)** – містить повний перелік команд, що керують процесом розробки програми.

**Панель інструментів (2)** – кнопки для виконання найпоширеніших команд.

**Панель елементів (3)** – містить основні візуальні та невізуальні компоненти, які програміст може включати у додаток на етапі розробки. Ці операції не вимагають програмування, необхідний код програми генерується автоматичною

**Редактор форм (4)** – (форма, це вікно яке використовується у додатку та містить свої власні компоненти) Редактор форм дозволяє створювати вікно необхідного вигляду розміщуючи на ньому необхідні компоненти.

**Редактор коду (5)** – вікно призначене для внесення змін у вихідний текст автоматично згенерованої програми.

**Панель «Свойства» (6)** - відображає властивості вибраного (*виділеного*) в даний момент об'єкту.

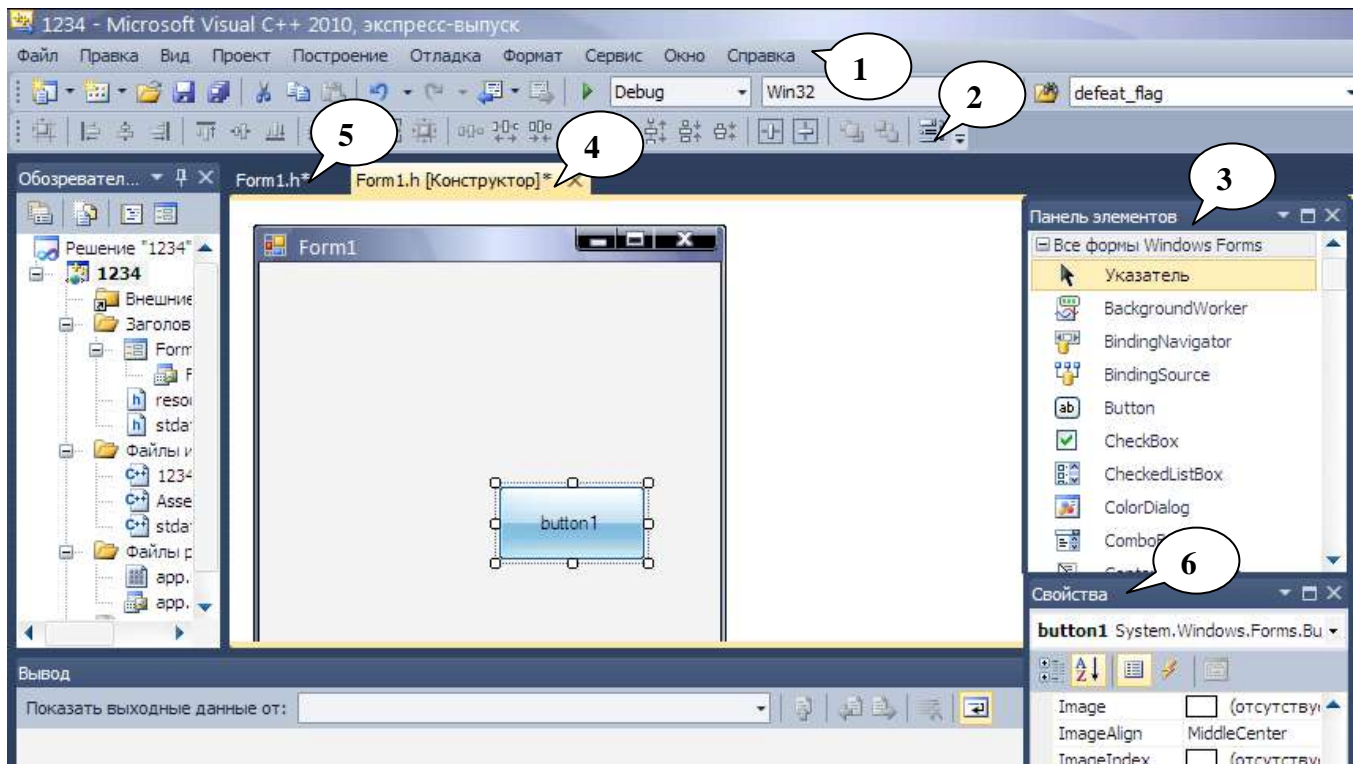


Рис.1. Интерфейс среды

У верхній частині вікна знаходиться рядок меню і область відображення панелей інструментів. За умовчанням у області відображення панелей інструментів виводиться панель «Стандартная». Щоб зробити доступними інші панелі інструментів, треба вибрати команду «Вид > Панель инструментов».

Центральну частину вікна Visual C++ займає вікно редактора форми (4). У ньому знаходиться форма - заготовка вікна додатку (вікно програми під час його розробки прийняте називати формою). У вікні дизайнера форми у графічній формі відображаються інструкції програми (у заголовку вікна вказане ім'я h-файлу), що забезпечують створення вікна. Щоб побачити форму, виберіть в меню «Вид» команду «Конструктор». Щоб переглянути код програми «Вид > Код».

У правій частині головного вікна Visual C++ зазвичай відображаються вікна «Панель элементов» і «Свойства» (рис.2).

У вікні «Панель элементов» знаходяться компоненти, які можна помістити (перетягнути) на форму. Компонент - це об'єкт, що реалізовує деяку функціональність.

У вікні «Свойства» відображаються властивості вибраного (виділеного) в даний момент об'єкту - компоненту або, якщо жоден з елементів не виділений, самої форми. Ім'я і тип об'єкту відображаються у верхній частині вікна «Свойства».

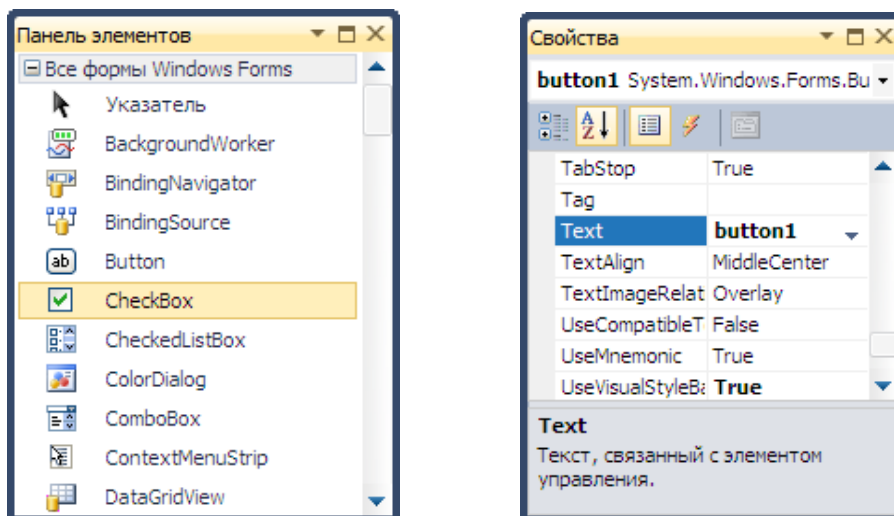


Рис.2. Вікна «Панель элементов» та «Свойства»

Вікно «Свойства» використовується для редагування значень властивостей об'єктів і, як наслідок, зміни їх вигляду і поведінки. Наприклад, щоб змінити надпис на кнопці (**button**), треба змінити значення властивості **Text**.

При відображенні властивості можуть бути об'єднані в групи за функціональною ознакою «По категориям» або впорядковані за абеткою «В алфавитном порядке» (рис.3).

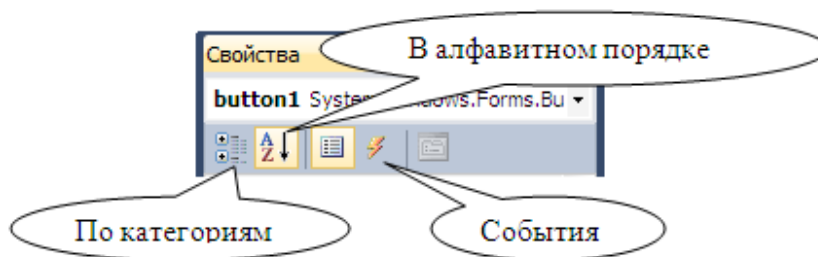


Рис.3 Панель інструментів вікна «Свойства»

Натиснувши у вікні «Свойства» кнопку «События», можна побачити події, які здатний сприймати вибраний об'єкт (рис.4). **Подія** - це те, що відбувається під час роботи програми. Наприклад, командна кнопка може реагувати на клацання кнопкою миші - подія «Click».

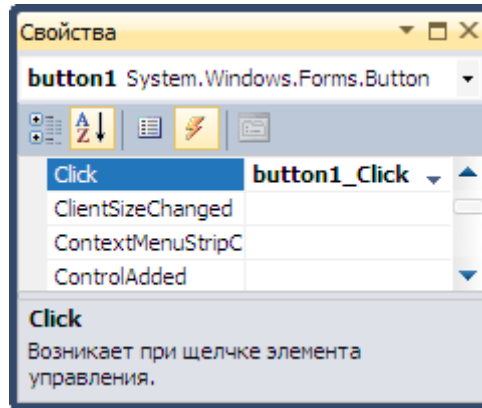


Рис.4. Вікно «Свойства» вкладка «События»

У вікні «Обозреватель решений» відображається структура проекту. У ньому перераховані файли, створюючи проект. Функція «main» знаходиться в головному файлі проекту (\*.cpp), опис класу форми - в h-файлі (Form1.h). Тут же відображаються імена файлів ресурсів (\*.rc) і значка (\*.ico) проекту та ін. (рис.5).

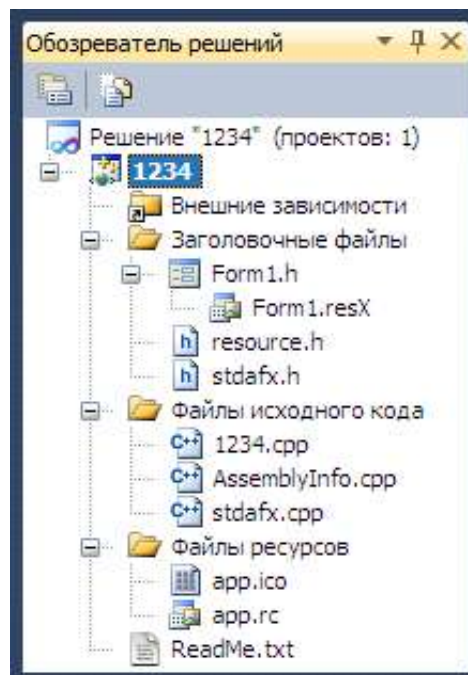


Рис.5. Вікно «Обозреватель решений»

Вікно «Обозреватель решений» зручно використовувати для швидкого доступу до необхідного елемента проекту.

## Етапи розробки додатків

На початку створення проекту необхідно виконати команду «Файл > Создать > Проект».

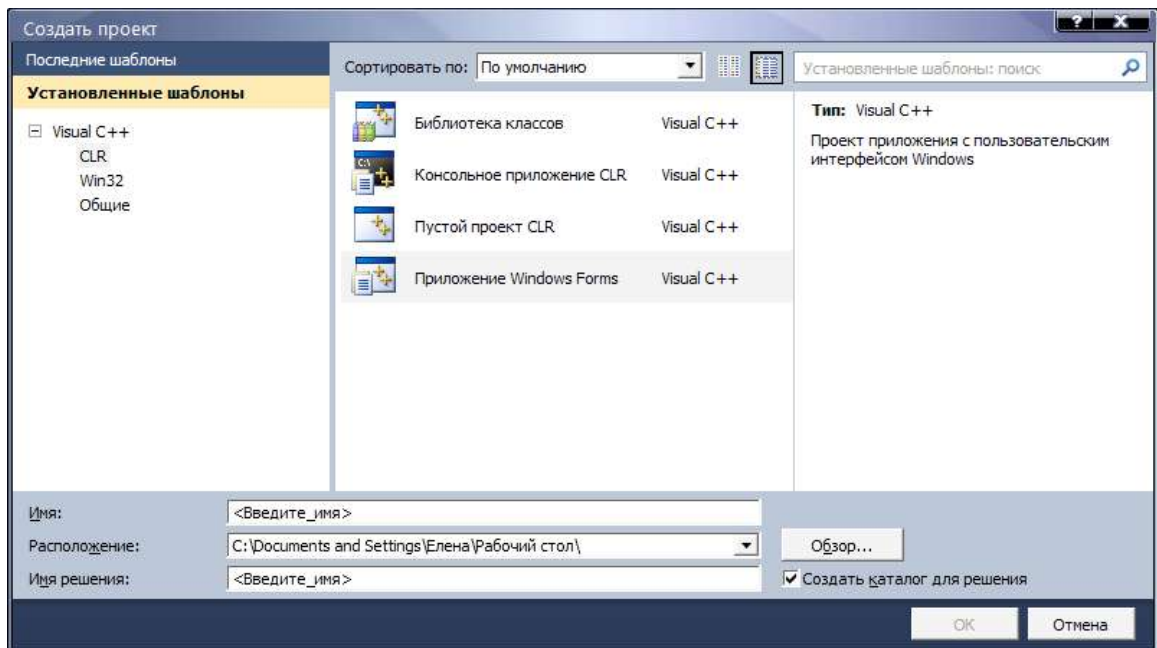


Рис.6. Створення проекту

У вузлі Visual C++ області встановлених шаблонів необхідно вибрати **CLR**, а потім у області шаблонів вибрати шаблон «**Приложение Windows Forms (Visual C++)**». Після чого необхідно ввести ім'я проекту в поле «**Имя**» і шлях до файлів проекту (рис.6).

У вікні (рис.1) відображається екранна форма «**Form1**». Форма відобразиться у вкладці «**Form1.h [Конструктор]**». У формі розташовують компоненти графічного інтерфейсу користувача (елементи управління). Це поля для введення тексту «**TextBox**», командні кнопки «**Button**», мітки «**Label**» - які не можуть бути відредаговані користувачем, і інші елементи управління. Візуальне програмування, передбачає просте перетягування елементів за допомогою миші з «**Панели элементов**», де розташовані всі елементи управління. Це допомагає звести до мінімуму безпосереднє написання програмного коду.

Властивостями кнопки «**button1**» є: ім'я кнопки «**Name**», напис на кнопці «**Text**», розташування кнопки «**Location**» в системі координат форми X, Y, розмір кнопки «**Size**», активність «**Enabled**» і видимість «**Visible**» та ін.

Властивості можна побачити, якщо клацнути правою кнопкою миші в межах форми і вибрати у контекстному меню команду «Свойства», при цьому з'явиться панель властивостей (якщо вона не відображається на екрані).

У випадку, якщо зникають панелі елементів чи властивостей, їх можна відобразити на екрані використовуючи кнопки на панелі інструментів (рис. 7).

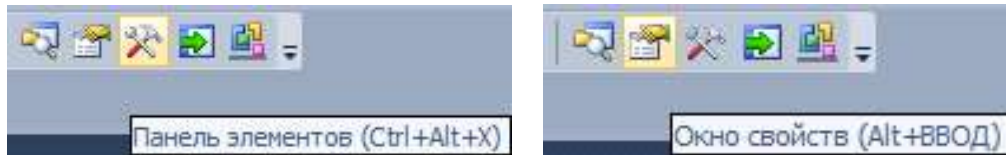


Рис.7. Фрагмент панелі інструментів

Об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці «Click», клацання в межах форми, завантаження форми «Load» в оперативну пам'ять при старті програми.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у властивостях елемента клацнути на значку блискавки «События» і в списку всіх можливих подій вибрати подвійним клацанням необхідна подія. Після цього активізується вкладка програмного коду «Form1.h».

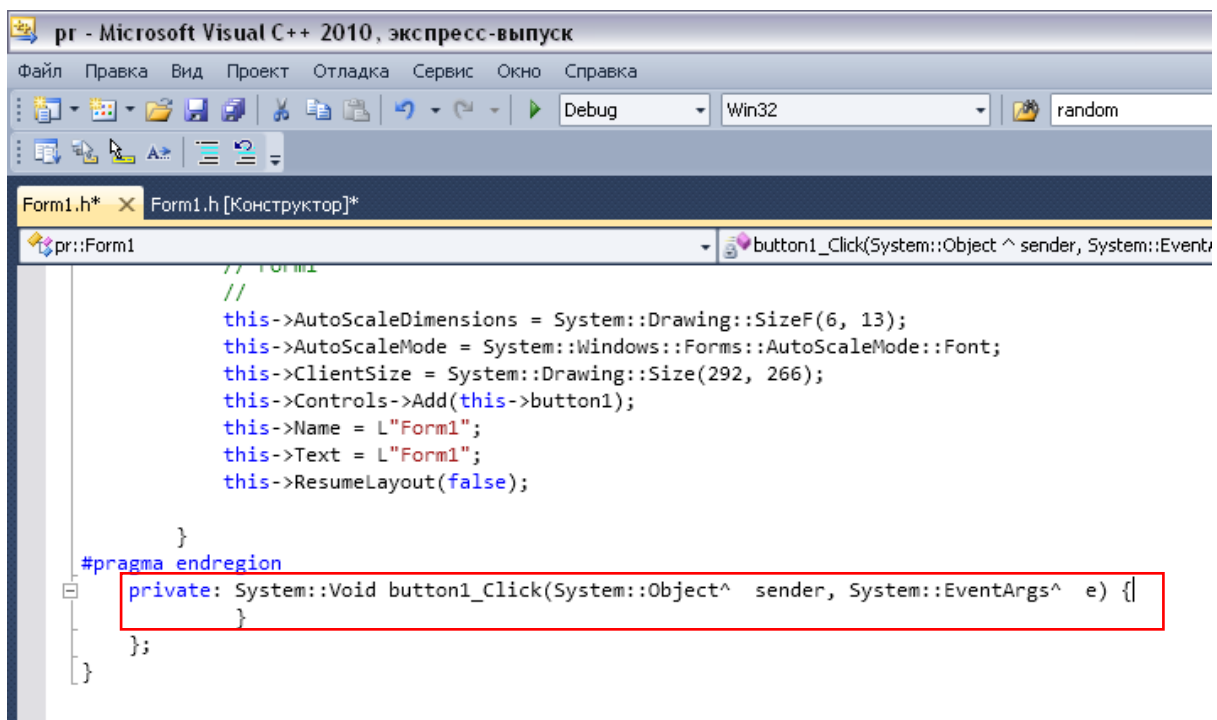


Рис.8. Обробник подій

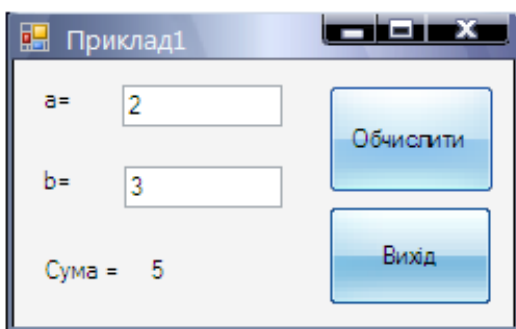
Перемикається між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш «**Ctrl+Tab**» (як це прийнято звичайно при перемиканні між вкладками в Windows).



Рис.9. Запуск проекту

Запуск проекту здійснюється командою «**Отладка > Начать отладку**», або натиснути клавішу **F5** (рис.9).

### Приклад програми обчислення суми двох чисел



```
private: System::Void button1_Click(...)  
{  
    double a,s,b;  
    a=Convert::ToDouble(textBox1->Text);  
    b=Convert::ToDouble(textBox2->Text);  
    s=a+b;  
    label1->Text=Convert::ToString(s);  
}
```

**//Convert::ToDouble** – перетворення змінної до дійсного типу

**//Convert::ToString** - перетворення змінної до рядкового типу

Для виведення повідомлень у окремому вікні можна використати метод **Show**.

```
MessageBox::Show("Текст");
```

Тут викликається метод **Show** об'єкту **MessageBox** з текстом. Оператор розширення області дії (::) вказує системі знайти метод **Show** серед методів об'єкту **MessageBox**. Таким чином, об'єкти окрім властивостей мають також і методи, тобто програми, які обробляють об'єкти.

## Можливості відлагодження Windows-додатків

Успішне завершення процесу побудови не означає, що в програмі немає помилок. Переконайтеся, що програма працює правильно, можна тільки в процесі перевірки її працездатності- тобто *тестуванням*.

Якщо програма працює коректно тільки на деякому обмеженому наборі початкових даних, це свідчить про те, що в програмі є алгоритмічні помилки. Процес пошуку і усунення помилок називається *відлагодженням*.

### Класифікація помилок

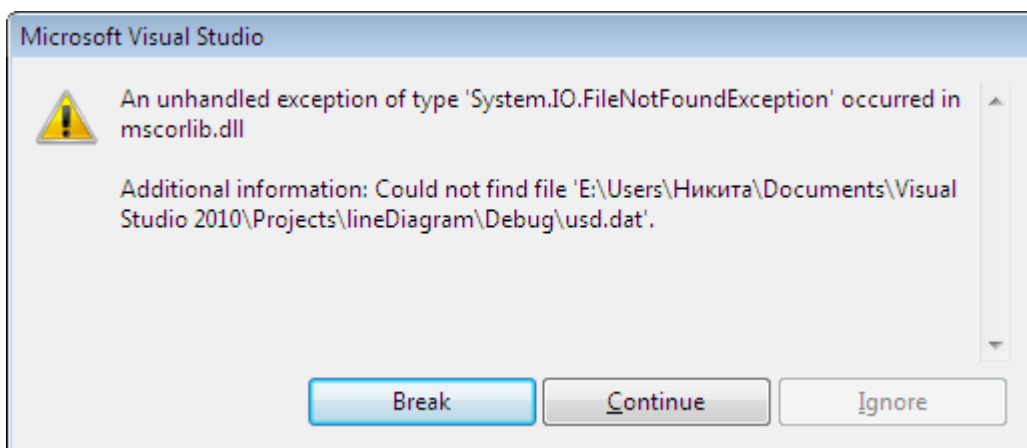
Помилки, які можуть бути в програмі, прийнято ділити на три групи:

- синтаксичні;
- помилки часу виконання;
- алгоритмічні.

*Синтаксичні помилки*, їх також називають *помилками часу компіляції* (**Compile-time error**). Їх виявляє компілятор, а програмісту залишається тільки внести зміни в текст програми і виконати повторну компіляцію.

*Помилки часу виконання*, вони називаються виключеннями (**exception**), легко виправляються. Вони звичайно виявляються вже при перших запусках програми і під час тестування.

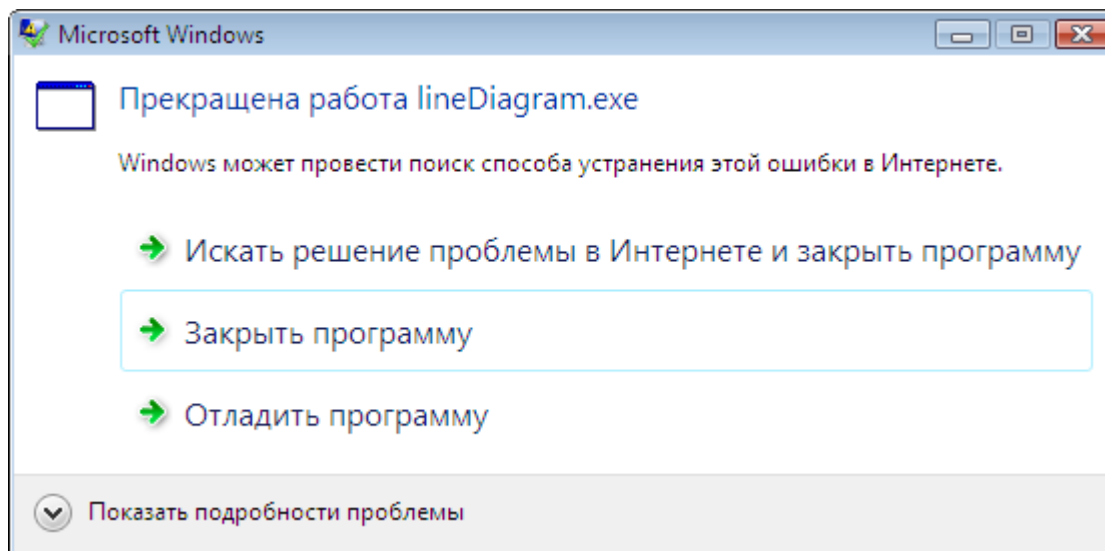
При виникненні помилки (виключення) в програмі, запущеній з середовища розробки, на екрані з'являється вікно, в якому відображається інформація про тип (класі) виключення і інформаційне повідомлення, що пояснює причину виникнення виключення.





*Приклад повідомлення про помилку, причина якої - відсутність (недоступність) файлу, потрібного програмі.*

Якщо програма запущена з операційної системи, то при виникненні виключення на екрані також з'являється повідомлення про помилку, але тип помилки в повідомленні не вказується.



Компілятор виявити *алгоритмічні помилки* не може. Тому, якщо в програмі є алгоритмічні помилки, компіляція завершується успішно. Переконавшись в тому, що програма працює правильно і в ній немає алгоритмічних помилок, можна тільки в процесі *тестування* програми. Процес пошуку алгоритмічних помилок може бути трудомістким, що вимагає аналізувати алгоритм, вручну "прокручувати" процес його виконання.

### **Запобігання і обробка помилок**

У програмі під час її роботи можуть виникати виключення (помилки), зокрема внаслідок дій користувача. *Наприклад, може ввести невірні дані або, що буває досить часто, видалити потрібний програмі файл.*

Порушення в роботі програми називається *виключенням*. Стандартну обробку виключень, яка в загальному випадку полягає у відображенні повідомлення про помилку, бере на себе код, що автоматично додається у виконувану програму. Разом з тим програміст може (і повинен) забезпечити явну обробку виключень. Для цього

в текст програми необхідно помістити інструкції, що забезпечують обробку можливих виключень.

Інструкція обробки виключення в загальному вигляді виглядає так:

**try**

```
{ /*фрагмент програми у якій може виникнути непередбачувана ситуація*/ }
```

**catch** (оголошення виключення)

```
{ /*оператори для обробки виключення, яке виникло в try-блоці*/ }
```

**throw** вираз

**try**

```
{ //здесь інструкції, виконання которых может вызвать исключение }
```

```
// начало секции обработки исключений
```

**catch** (ExceptionType\_1^ e)

```
{ //инструкції обробки исключения ExceptionType_1 }
```

**catch** (ExceptionType\_2^ e)

```
{ //инструкції обробки исключения ExceptionType_2 }
```

**catch** (ExceptionType\_i^ e)

```
{ //инструкції обробки исключения ExceptionType_i }
```

Здесь:

- **try** — ключове слово, що показує, що далі слідують інструкції, при виконанні яких можливо виникнення виключень, і що обробку цих виключень бере на себе програма;

- **catch** — ключове слово, що позначає початок секції обробки виключення вказаного типу. Інструкції цієї секції будуть виконані при виникненні виключення вказаного типу.

- **throw** (кинути) - ключове слово, що "створює" виключення.

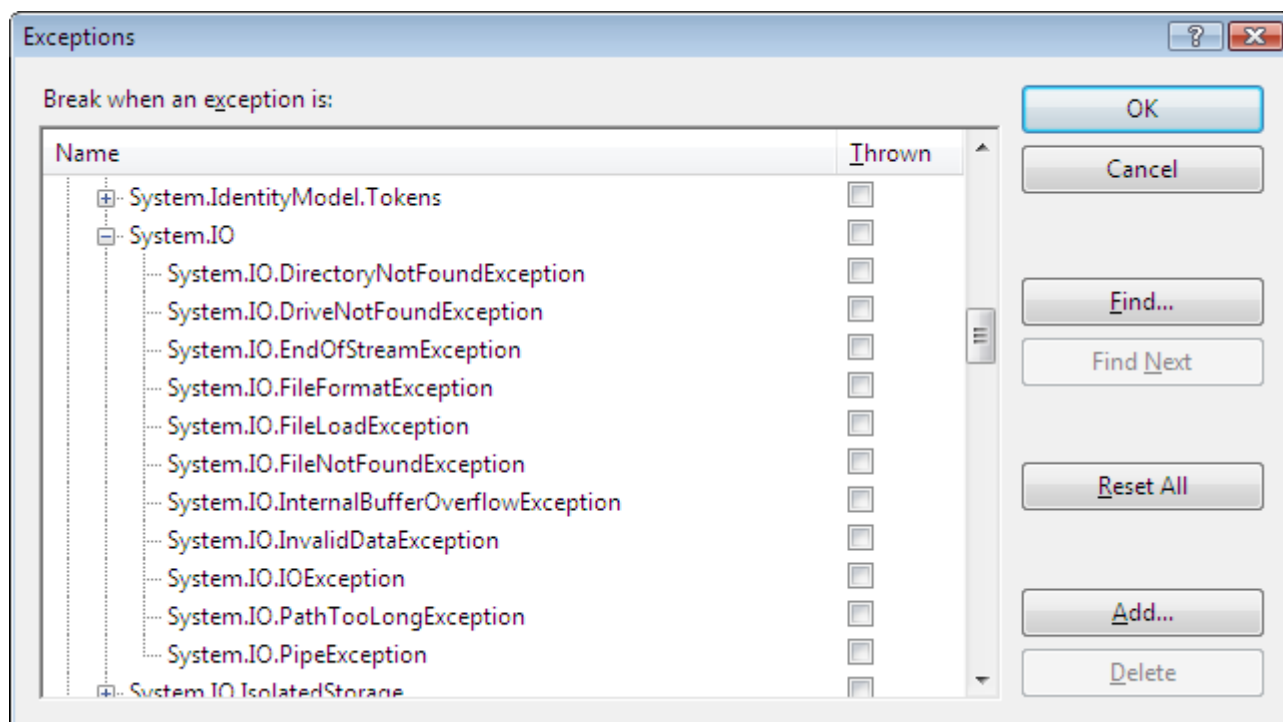
Основною характеристикою виключення є його тип.

## Виключення і причини, які можуть привести до їх виникнення.

Тип виключення	Виключення, причина
System::FormatException	Помилка формату. Виникає при виконанні перетворення, якщо перетворювана величина не може бути приведена до необхідного вигляду. Найчастіше виникає при спробі перетворити рядок символів в число, якщо рядок містить невірні символи. <i>Наприклад, при перетворенні рядка в дробове значення, якщо як десятковий роздільник замість коми поставлена крапка</i>
System::IndexOutOfRangeException	Вихід значення індексу за межу області допустимого значення. Виникає при зверненні до неіснуючого елемента масиву
System::IO::FileNotFoundException	Немає файлу. Причина - відсутність необхідного файлу у вказаному каталозі
System::IO::DirectoryNotFoundException	Немає каталогу. Причина - відсутність необхідного каталогу

Інші виключення можна побачити, вибравши в меню **Debug** команду **Exceptions** і у вікні, що з'явилося, розкривши відповідну групу.

**Виключення, які можуть виникнути при виконанні операцій файлового введення/виведення.**



## ТЕМА 2. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ

### Особливості роботи з формами

Форми є основою додатків. Створення інтерфейсу користувача додатку полягає в додаванні у вікно форми об'єктів C++, які називаються компонентами. Компоненти розташовуються на панелі елементів, виконаній у вигляді розгортаючогося списку.

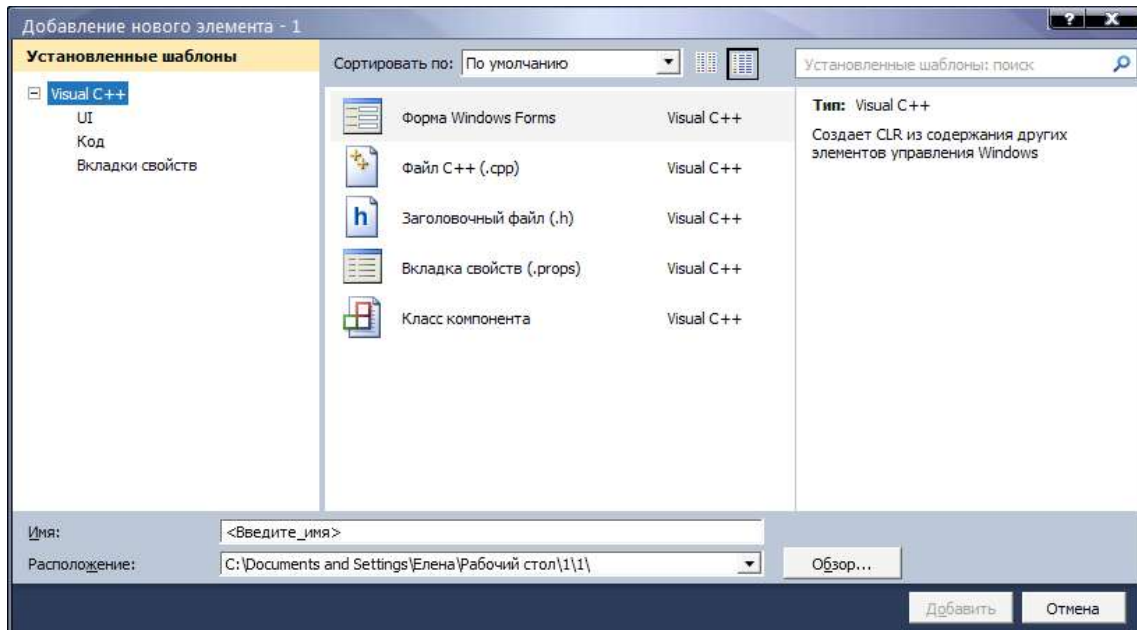


Рис.10. Додавання нової форми

Для додавання у проект нової форми необхідно обрати команду «**Проект > Додати новий елемент...**». У діалоговому вікні обрати «**Форма Windows Forms**» та задати ім'я форми. У результаті з'явиться вкладка ім'я.h [Конструктор] поряд з вкладкою конструктора головної форми **Form1.h [Конструктор]**.

Форма, що з'являється у проекті при його створенні завантажується першою за замовчуванням, тобто завжди вважається стартовою (головною).

Якщо у проект додано форму її необхідно викликати відповідним методом.

Форма викликається на виконання у двох режимах: модальному та немодальному. У модальному режимі – метод форми **ShowDialog()**, у звичайному – методом форми **Show()**.

Форма має велику кількість методів, зокрема:

**Close()** – закрити форму. Для головної форми, це означає закриття додатку.

**Hide()** – приховати форму (зробити невидимою)

**Show()** – виведення форми на екран

Якщо формами буде використано один елемент, наприклад з метою передачі даних елемент **"textBox1"** буде знаходитися на обох формах, то його потрібно оголосити як **public** у верхній частині коду форми **"form2"**, а в кодї форми **"Form1"** в самому верху коду підключити бібліотеку другої форми (*розділ #pragma once*) **#include "form2.h"**, так само в одній з подій потрібно писати:

```
form2^ gform2 = gcnew form2;
```

```
gform2->Show();
```

```
gform2-> textBox1->Text = this->textBox1->Text;
```

## **Характеристика основних компонентів середовища**

Компоненти розділяються на видимі (візуальні) і невидимі (невізуальні) (*рис.11*). Візуальні компоненти з'являються під час виконання точно так, як і під час проектування. Прикладами є кнопки і поля редагування. Невізуальні компоненти з'являються під час проектування як піктограми на формі. Вони не видні під час виконання, але мають певне функціональне навантаження (*наприклад, забезпечують доступ до даних, викликають стандартні діалоги Windows*).

Для додавання компоненту у форму можна вибрати мишею потрібний компонент в палітрі і клацнути лівою клавішею миші в потрібному місці проектованої форми. Компонент з'явиться на формі, і далі його можна переміщати та встановлювати характеристики.

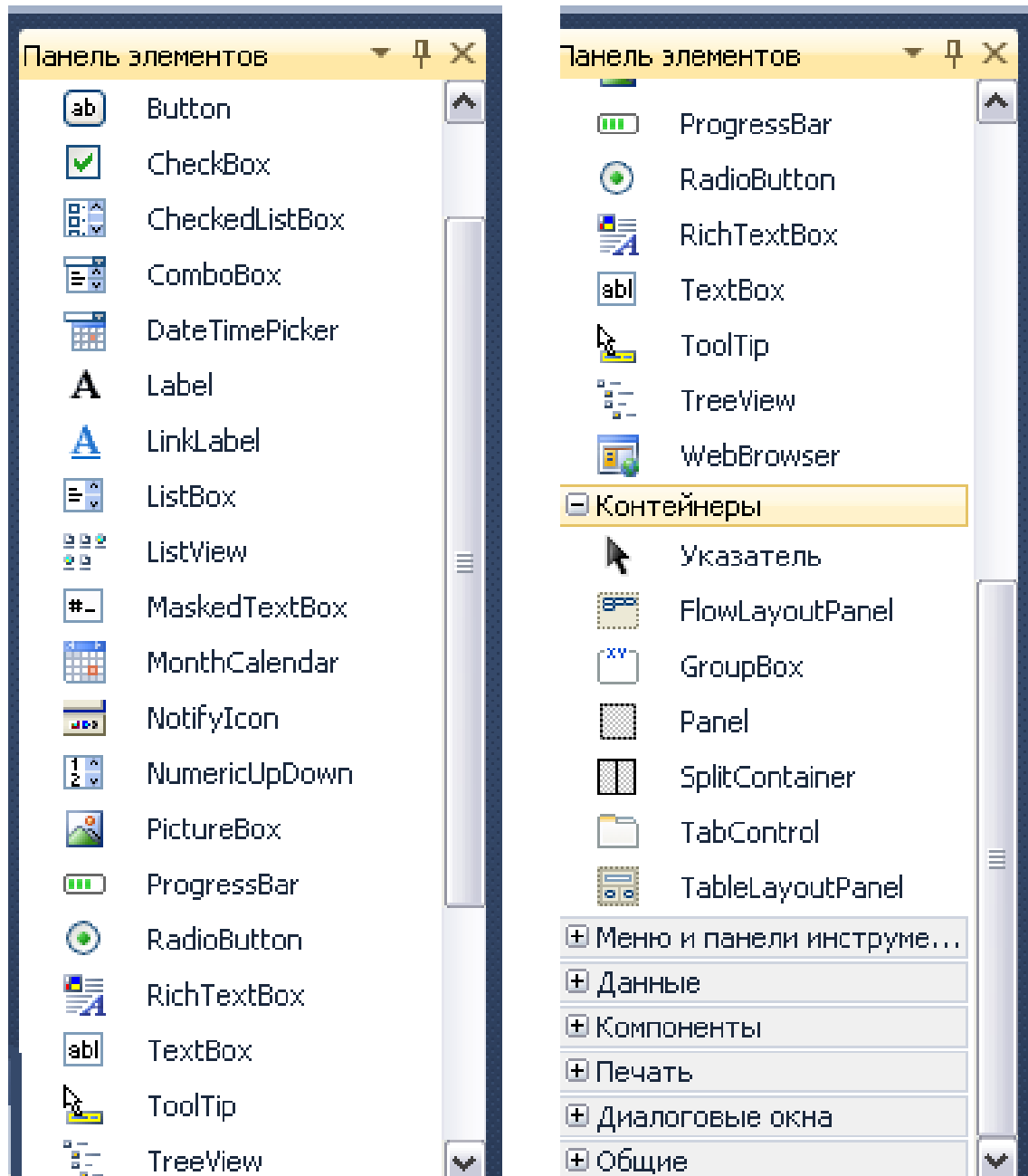


Рис.11. Панель елементів

Кожен компонент має три різновиди характеристик: **властивості, події і методи.**

**Властивості** є атрибутами компоненту, що визначають його зовнішній вигляд і поведінку. Багато властивостей компоненту мають значення, що встановлюється за замовчуванням (наприклад, висота кнопок). Властивості компоненту відображаються а сторінці властивостей Можна визначити властивості під час проектування або написати код для видозміни властивостей компоненту під час

виконання додатку (*видимий чи не видимий компонент в залежності від виконання умови*).

Сторінка **подій** показує список подій, розпізнаваних компонентом (*програмування для операційних систем з графічним інтерфейсом користувача*). Кожен компонент має свій власний набір обробників подій. Створюючи обробник події, ви доручаєте програмі виконати написану функцію, якщо ця подія відбудеться.

**Метод** - є функцією, яка пов'язана з компонентом, і яка оголошується як частина об'єкту. Створюючи обробники подій, можна викликати методи, використовуючи наступну нотацію: `->`.

**Наприклад:** `button1-> Hide()` //приховати кнопку.

У бібліотеці візуальних компонентів існує множина компонентів, що дозволяють відображати, вводити та редагувати текстову інформацію



## Компонент Button

Компонент **Button** є командною кнопкою.

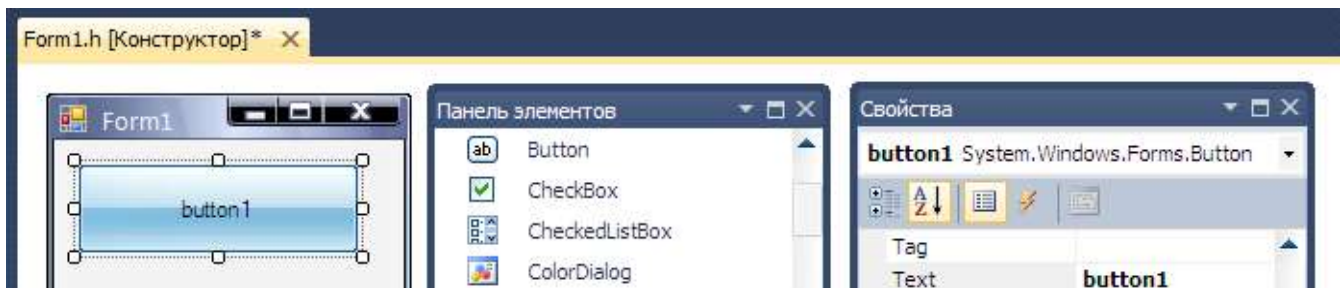


Рис.12. Компонент **Button**

Кожен компонент має визначений набір властивостей методів та подій.

Таблиця 1. Характеристики компоненту **Button**

<b>Властивості компоненту Button</b>	
<b>Name</b>	Ім'я компоненту. Використовується в програмі для доступу до властивостей компоненту
<b>Text</b>	Текст (напис) на кнопці
<b>TextAlign</b>	Положення тексту на кнопці. Напис може розташовуватися в центрі ( <b>MiddleCenter</b> ), бути притиснута до лівої ( <b>MiddleLeft</b> ) або правої ( <b>MiddleRight</b> ) межі.
<b>Location</b>	Положення кнопки на поверхні форми.
<b>Size</b>	Розмір кнопки
<b>Font</b>	Шрифт тексту
<b>ForeColor</b>	Колір тексту
<b>Enabled</b>	Ознака доступності кнопки. Кнопка доступна, якщо значення властивості рівне <b>True</b> , і недоступна ( <i>наприклад, подія <b>Click</b> на кнопці не виникає</i> ), якщо значення властивості рівне <b>False</b>
<b>Visible</b>	Дозволяє приховати кнопку ( <b>False</b> ) або зробити її видимою ( <b>True</b> )
<b>Image</b>	Картинка на поверхні форми. Рекомендується використовувати gif-файл, в якому визначений прозорий колір фону
<b>ImageAlign</b>	Положення картинки на кнопці
<b>Anchor</b>	Закріплення позиції компоненту
<b>DialogResult</b>	Забезпечує роботу з модальними формами
<b>TabStop</b>	Відключення отримання фокуса за допомогою клавіші Tab (=False)
<b>Події компоненту Button</b>	
<b>Click</b>	Виникає, коли на кнопці клікають мишею
<b>Enter</b>	Виникає, коли кнопка отримує фокус, тобто кнопка стає активною
<b>Методи компоненту Button</b>	
<b>Hide()</b>	Приховує кнопку
<b>Focus()</b>	Робить кнопку активною
<b>Show()</b>	Відображає кнопку



## Компонент Label

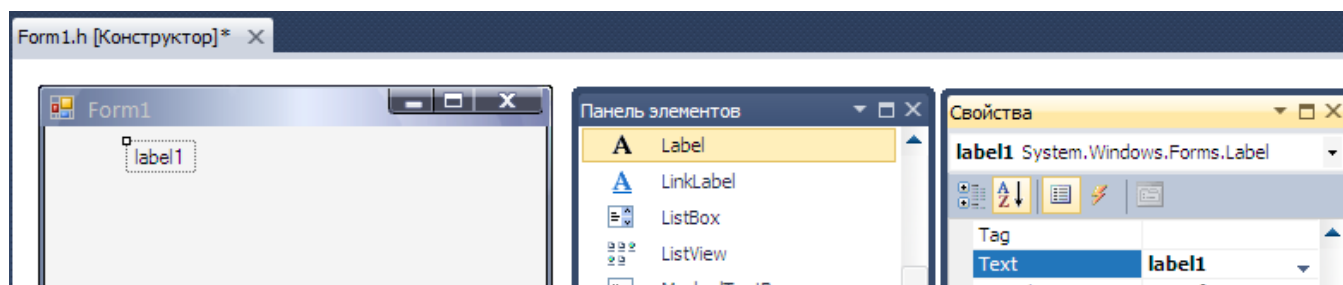


Рис.13. Компонент **Label**

Компонент **Label** призначений лише для відображення текстової інформації (редагування тексту не можливо). Задати текст, що відображається в полі компоненту, можна як під час розробки форми, так і під час роботи програми, присвоївши значення властивості **Text**.

Компонент може бути використаний також для виведення зображень та ідентифікації об'єктів.

Таблиця 2. Властивості компоненту **Label**

<b>Name</b>	Ім'я компоненту. Використовується в програмі для доступу до властивостей компоненту
<b>Text</b>	Текст (напис) у полі
<b>TextAlign</b>	Положення тексту у полі компоненту (вирівнювання)
<b>Location</b>	Положення компоненту на поверхні форми
<b>Size</b>	Розмір компоненту
<b>Font</b>	Шрифт тексту
<b>ForeColor</b>	Колір тексту
<b>BackColor</b>	Колір фону
<b>BorderStyle</b>	Вид рамки (межі) компоненту

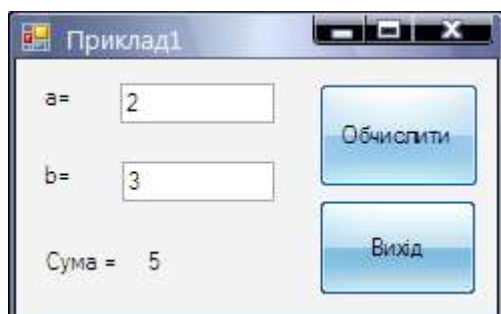
Щоб в полі компоненту **Label** вивести числове значення, це значення необхідно перетворити в рядок. Зробити це можна за допомогою методу **ToString**.

**Convert::ToString( s )** - перетворення змінної до рядкового типу

Колір тексту «**ForeColor**» і фону «**BackColor**» можна задати, вказавши назву кольори (**Color::Red**, **Color::Blue**, **Color::Green** і т. д.), або елемент колірної схеми операційної системи. У другому випадку колір буде "прив'язаний" до поточної

колірної схеми операційної системи і автоматично змінюватиметься при кожній її зміні. За замовчанням для елементів управління використовується другий спосіб кодування кольору. Колір фону може бути "прозорим" (**Color::Transparent**).

### Приклад програми обчислення суми двох чисел



```
private: System::Void button1_Click(...)
{
    double a,s,b;
    a=Convert::ToDouble(textBox1->Text);
    b=Convert::ToDouble(textBox2->Text);
    s=a+b;
    label1->Text=Convert::ToString(s);
}
```

//**Convert::ToDouble** – перетворення змінної до дійсного типу

//**Convert::ToString** - перетворення змінної до рядкового типу



## Компонент **TextBox**

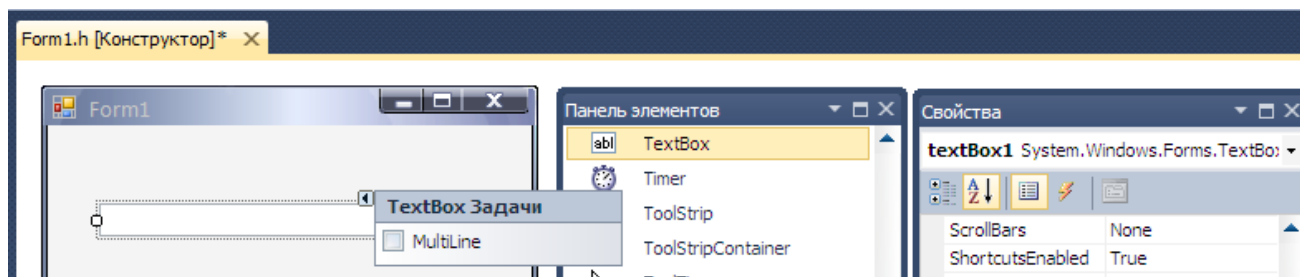


Рис.14. Компонент **TextBox**

Компонент **TextBox** призначений для введення даних з клавіатури. Залежно від налаштування компоненту, в полі редагування можна вводити один або декілька (**Multiline = True**) рядків тексту.

Всі дані ведені у текстове поле сприймаються програмою, як набір символів. Для перетворення символів до дійсного типу використовують методи **ToDouble** або **ToInt32** (цілочисловий тип).

**Convert::ToDouble** – перетворення змінної до дійсного типу

Таблиця 3. Властивості компоненту **TextBox**

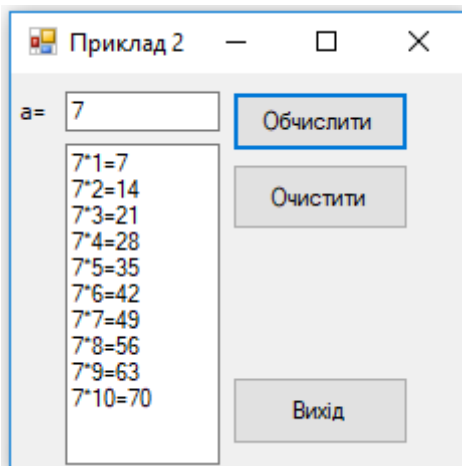
<b>Name</b>	Ім'я компоненту. Використовується в програмі для доступу до компоненту і його властивостей, зокрема до тексту, який знаходиться в полі редагування
<b>Text</b>	Текст, який знаходиться в полі редагування
<b>Location</b>	Положення компоненту на поверхні форми
<b>Size</b>	Розмір компоненту
<b>Font</b>	Шрифт, використаний для відображення тексту в полі компоненту
<b>ForeColor</b>	Колір тексту
<b>BackColor</b>	Колір фону поля компоненту
<b>BorderStyle</b>	Вид рамки (межі) компоненту. Межа компоненту може бути звичайною ( <b>Fixed3D</b> ), тонкою ( <b>FixedSingle</b> ). Межа навколо компоненту може бути відсутньою (в цьому випадку значення властивості рівне <b>None</b> )
<b>TextAlign</b>	Спосіб вирівнювання тексту в полі компоненту.
<b>MaxLength</b>	Максимальна кількість символів, яку можна ввести в поле компоненту
<b>PasswordChar</b>	Символ, який використовується для відображення символів, що вводяться користувачем ( <i>введений користувачем рядок знаходиться у властивості Text</i> )
<b>Multiline</b>	Вирішує ( <b>True</b> ) або забороняє ( <b>False</b> ) введення декількох рядків тексту
<b>Lines</b>	Масив рядків, елементи якого містять текст, що знаходиться в полі редагування, якщо компонент знаходиться в режимі <b>MultiLine</b> . Доступ до рядка здійснюється по номеру. Рядки нумеруються з нуля
<b>SkroolBars</b>	Задає смуги прокрутки, що відображаються: <b>Horizontal</b> - горизонтальна; <b>Vertical</b> - вертикальна;

### Методи компоненту **TextBox**

<b>AppendText ()</b>	Додає текст до поточного тексту у вікні компоненту
<b>Clear ()</b>	Очищення поля
<b>Copy ()</b>	Копіювання обраного рядка у буфер обміну
<b>Cut ()</b>	Вирізання виділеного блоку тексту у буфер обміну

<b>Focus ( )</b>	Встановлення фокуса
<b>Hide ( )</b>	Приховування компонента
<b>Paste ( )</b>	Заміняє поточну вибірку в полі вмістом буферу обміну
<b>Select ( )</b>	Вибирає текст у компоненті
<b>Show ( )</b>	Робить компонент видимим

*Приклад програми виведення таблиці множення на введене з клавіатури число.*



```
//Кнопка «Обчислити»
private: System::Void button1_Click(...)
{
    double a, r;
    a=Convert::ToDouble(textBox1->Text);
    for(int i=1;i<=10;i++)
    {
        r=i*a;
        textBox2->AppendText(a+"*"+i+"="+r+"\r\n");
    }
}
```

```
// Кнопка «Очистити». Очистка полів форми.
private: System::Void button2_Click(...)
{
    textBox1->Clear();
    textBox2->Clear();
}
```

```
// Кнопка «Вихід» . Закриття додатку.
private: System::Void button3_Click(...)
{
    Close();
}
```



## Компоненти CheckBox і RadioButton

Радіокнопки утворюють групи взаємозв'язаних індикаторів, з яких звичайно може бути вибраний тільки один. Вони використовуються для вибору користувачем однієї з декількох взаємовиключних альтернатив

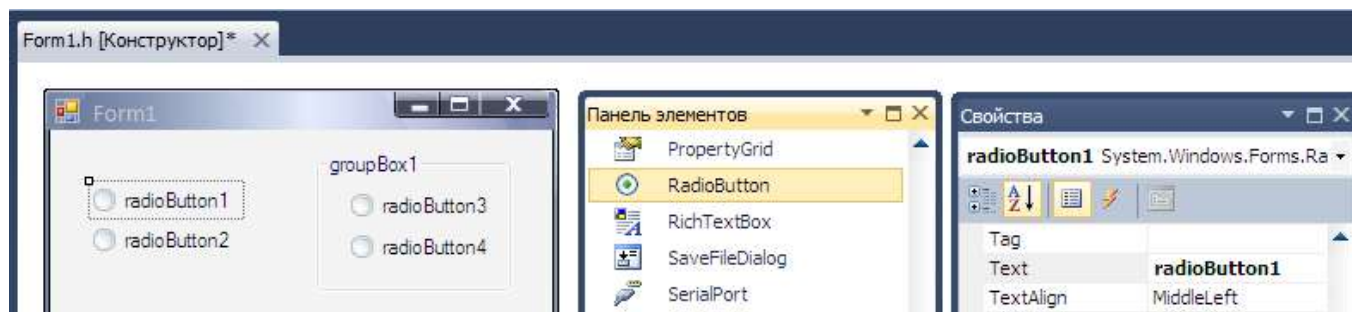


Рис.15. Компонент **RadioButton**

Окрема радіокнопка **RadioButton** особливого сенсу не має. А радіокнопки мають сенс, коли вони взаємодіють один з одним в групі.

Компонент **GroupBox** (вкладка «Контейнери» панелі елементів) – є контейнером, об'єднуючим групу зв'язаних елементів управління, таких, як радіокнопки **RadioButton** або індикатори **CheckBox** і т.д.

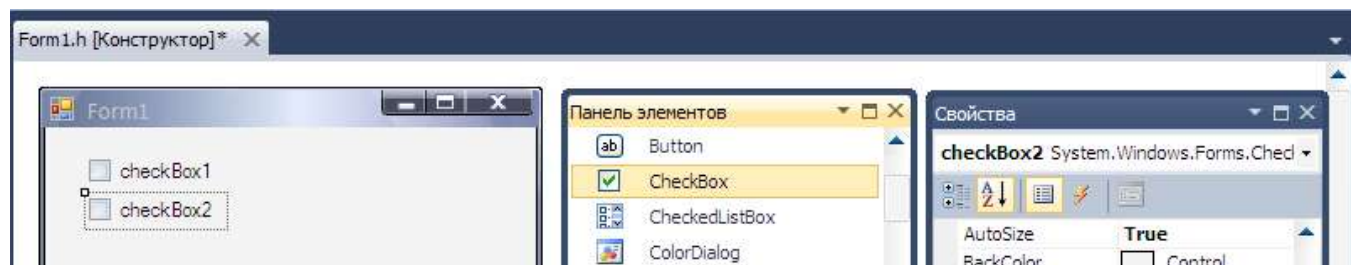


Рис.16. Компонент **CheckBox**

Індикатори з прапорцем **CheckBox** використовуються у додатках для того, щоб вмикати та вимикати опції, або для індикації стану. При кожному клацанні користувача на індикаторі його стан змінюється, проходячи в загальному випадку послідовно через три значення: виділення (поява чорної галочки), проміжне (сіре вікно індикатора і сіра галочка) і не виділене (порожнє вікно індикатора). Цим трьом станам відповідають три значення властивості компоненту.

### Основні властивості компоненту CheckBox

**Checked** – властивість, яка визначає чи включено прапорець.

**CheckState** – встановлює 3 види станів компоненту:

- **Checked** - Прапорець увімкнено;
- **Unchecked** - Прапорець вимкнено
- **Indeterminate** - Невизначений стан (сірий колір прапорця).

**CheckAlign** – властивість, що дозволяє відкривати розгортаючийся список, для вибору розташування прапорця у полі компонента (9 позицій).

**AutoCheck** - властивість визначає, чи повинно автоматично змінюватися стан прапорця в результаті клацання на його зображенні. За умовчанням значення рівне True.

Стан прапорця змінюється в результаті клацання на його зображенні (якщо значення властивості **AutoCheck = True**). При цьому виникає подія **CheckedChanged**, а потім подія **Click**. Якщо значення властивості **AutoCheck = False**, то в результаті клацання на прапорці виникає подія **Click**, а потім, якщо процедура обробки цієї події змінить стан кнопки, виникає подія **CheckedChanged**.

*Приклад програми обчислення суми або різниці двох чисел.*

```
private: System::Void
radioButton1_CheckedChanged(...)
{
    int a,b,s;
    a=Convert::ToInt32(textBox1->Text);
    b=Convert::ToInt32(textBox2->Text);
    if(radioButton1->Checked)
    {
        s=a+b;
        label1->Text="a+b =" +Convert::ToString(s);
    }
}
```

```
private: System::Void
radioButton2_CheckedChanged(...)
{
    int a,b,s;
    a=Convert::ToInt32(textBox1->Text);
    b=Convert::ToInt32(textBox2->Text);
    if(radioButton2->Checked)
    {
        s=a-b;
        label1->Text="a-b=" +Convert::ToString(s);
    }
}
```

## ТЕМА 3. РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА

### Компонент **MenuStrip**

Компонент **MenuStrip** є головним меню програми.

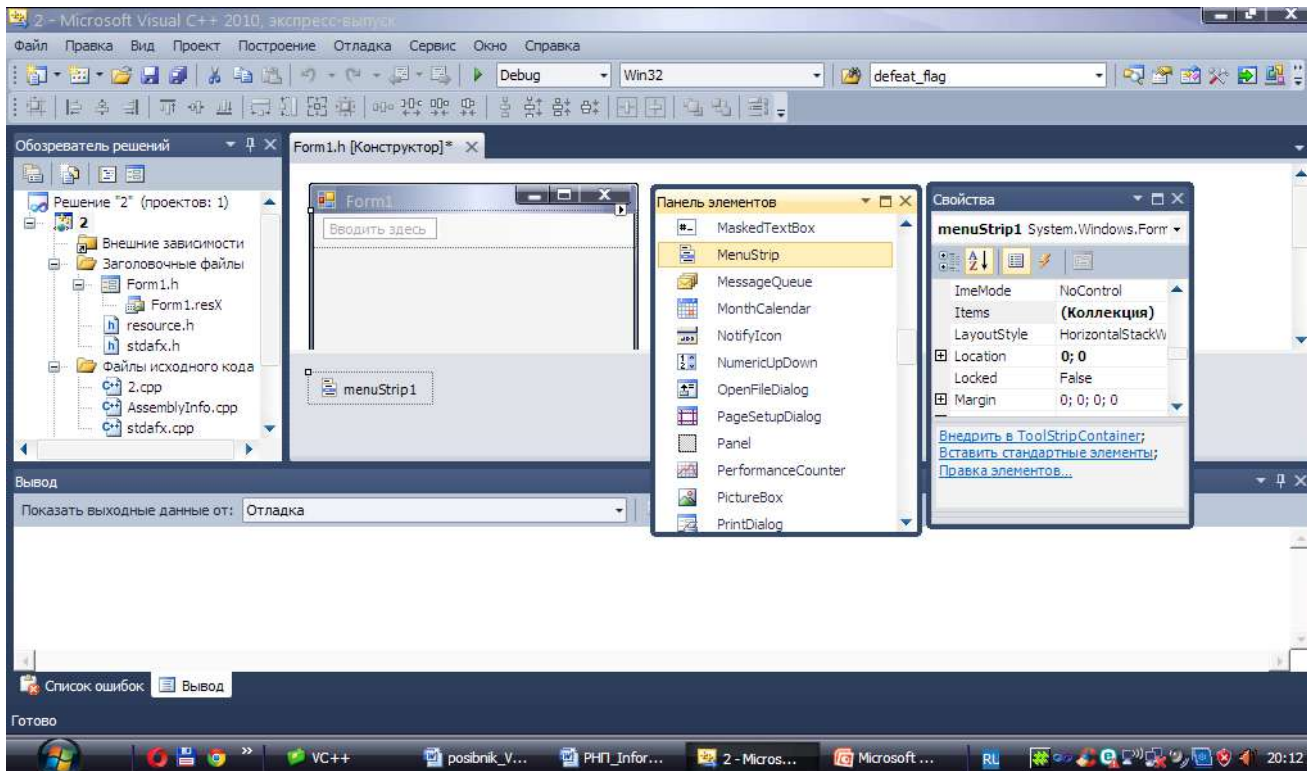


Рис.17. Компонент **MenuStrip**

Після того, як у форму буде доданий компонент **MenuStrip**, у верхній частині форми з'являється рядок меню, на початку якого знаходиться область введення тексту.

Окрім команди в меню можна додати роздільник - горизонтальну лінію. Роздільник звичайно служить для об'єднання в групи однотипних команд. Щоб додати його в список команд, слід виділити команду, перед якою треба помістити роздільник, і з контекстного меню вибрати команду «Вставка-Separator» (рис.18).

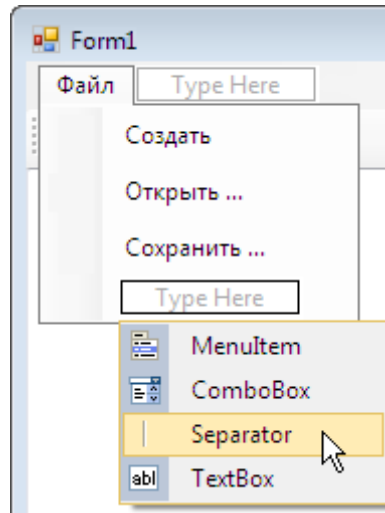


Рис.18. Створення пунктів меню

Після того, як структура меню буде сформована, можна виконати налаштування меню шляхом зміни значень властивостей пунктів меню (об'єктів типу **ToolStripMenuItem**).

#### Властивості об'єкту **ToolStripMenuItem**

**Text** - Назва елементу меню

**Image** - Картинка, яка відображається поряд з командою.

**Enabled** - Ознака доступності елементу меню.

**Checked** - Ознака того, що елемент меню вибраний. Якщо значення властивості = **True**, то елемент позначається галочкою. Властивість **Checked** звичайно застосовується для тих елементів меню, які використовуються для відображення параметрів, наприклад, пункт меню **Відобразити рядок стану**.

**ShortcutKeys** - властивість визначає комбінацію клавіш, натиснення якої активує виконання команди.

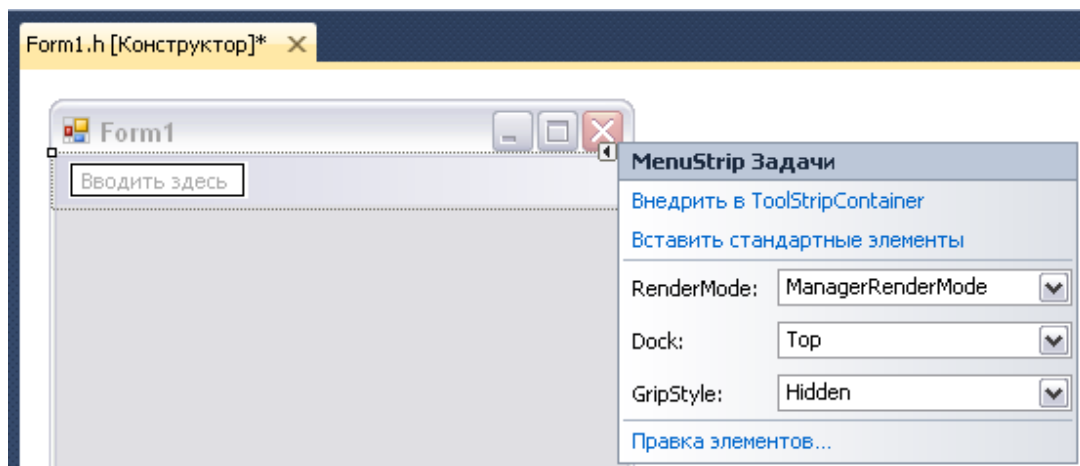


Рис.19. Налаштування головного меню



Діалогове вікно «**MenuStripЗадачи**» забезпечує доступ до типових команд та властивостей.

- **Внедрить ToolStripContainer** – дозволяє розмістити меню у спеціальний контейнер. Контейнер – це оберт з власним набором властивостей, встановлення яких дозволяє створювати зручне меню.
- **Внедрить стандартные элементы** – додає загальноприйняті опції меню.
- **RenderMode** – опція дає можливість вибору зі списку способу зображення меню (системне, професійне, управляєме).
- **Doc** – схема розташування меню відносно сторін форми.
- **GripStyle** – вибір зі списку стиля полоси меню (видима або невидима спеціальна пунктирна лінія)
- **Правка элементов** – встановлення опцій меню. Дана команда відкриває діалогове вікно для встановлення опцій меню, додавання нових опцій та їх реорганізація.

### Основні властивості компоненту **MenuStrip**

**BackgroundImage** – задає фонове зображення компоненту.

**Items** – дозволяє викликати діалогове вікно для налаштування опцій меню.

**LayoutStyle** - стиль розміщення меню.

**Checked** – властивість, яка визначає чи вибрано дану команду меню.

### Компонент **ContextMenuStrip**

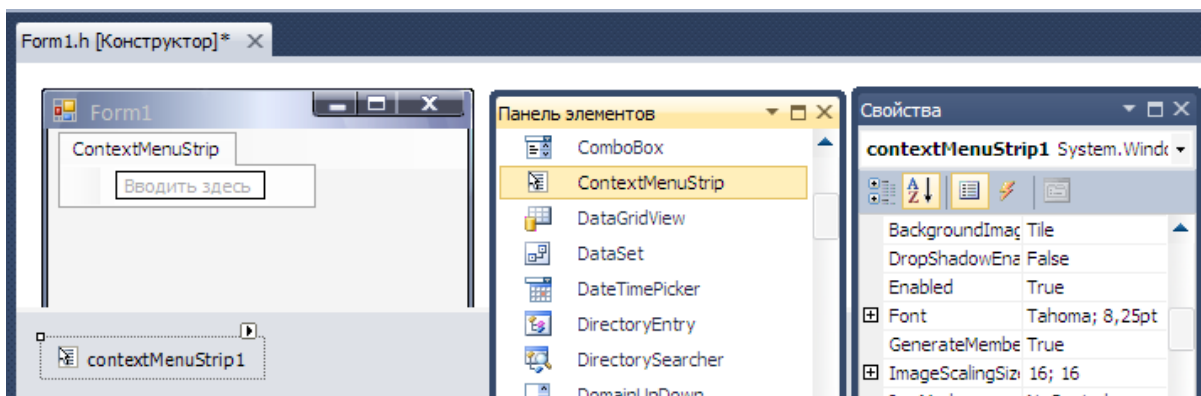


Рис.20. Компонент **ContextMenuStrip**

Даний компонент може бути прив'язаний до будь якого компоненту, який має властивість **ContextMenuStrip**. Коли компонент поміщають у форму його ім'я буде видно у будь якому компоненті форми. Встановлення опцій контекстного меню аналогічне головному меню.

## Компонент ToolStrip

Компонент **ToolStrip** - смуга (панель) інструментів використовується для розміщення в ньому інших компонентів. У панель інструментів можна помістити командну кнопку, поле відображення тексту, поле редагування, список, що розкривається.

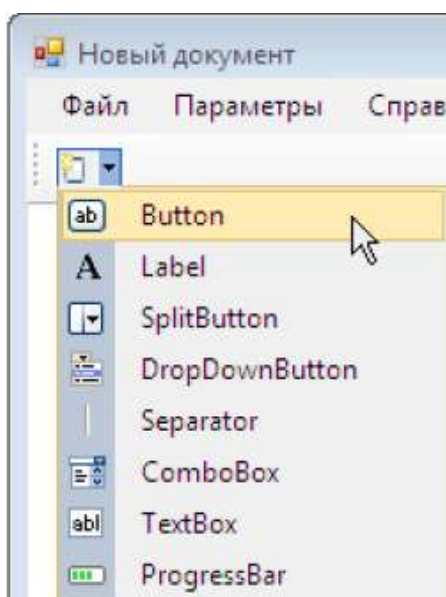


Рис.21. Створення панелі інструментів

### Властивості компоненту ToolStrip

**Buttons – (Items)** колекція компонентів, що знаходяться у панелі інструментів.

**Dock** – межа батьківського компоненту (форма) до якої прив'язано панель інструментів.

**Visible** – дозволяє приховати або відобразити панель інструментів.

**Enabled** – доступ до компонентів панелі інструментів.

Для того, щоб в панель інструментів додати, наприклад, командну кнопку, треба вибрати компонент **ToolStrip**, клікнути на значку розкриваючогося списку,

який відображається в його полі, і вибрати **Button**. Після цього треба виконати налаштування доданої кнопки **toolStripButton** - задати значення властивостей.

## Компонент StatusStrip

Компонент **StatusStrip** представляє собою область відображення службової інформації (рядок стану). Рядок стану відображається у нижній частині вікна додатку та зазвичай поділена на області. У цій області виводиться текстова інформація або індикатор процесу (**ProgressBar**) або командні кнопки.

Щоб у рядку стану відображався текст необхідно додати елемент **StatusLabel**. Необхідно обрати даний компонент у розгортаючому списку та налаштувати компонент **toolStripStatusLabel**.

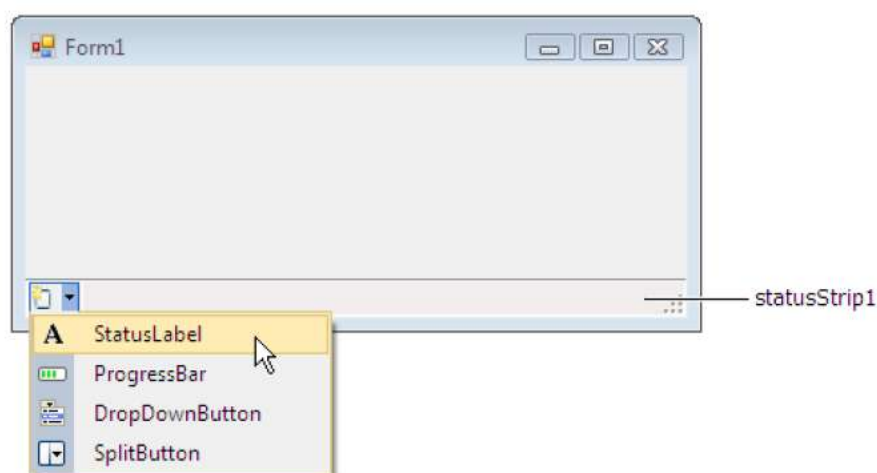


Рис.22. Компонент StatusStrip

*Наприклад. Виведення k-ті символів у полі введення*

```
private: System::Void textBox1_TextChanged(...)
{
    int len = textBox1->Text->Length;
    statusStrip1->Items[0]->Text = Convert::ToString(len);
}
```

## Стандартні діалогові вікна

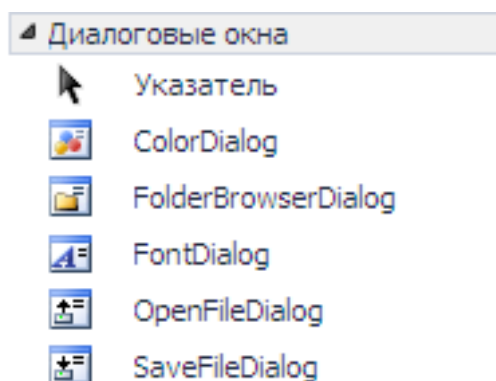


Рис.23. Діалогові вікна

### *Компонент OpenFileDialog*

Компонент представляє собою діалогове вікно «Открыть».

При додаванні на форму компонент розташовується у окремій області вікна дизайнера форм. Якщо необхідно відкрити папку, а не файл – використовують клас **FolderBrowserDialog**.

Діалогове вікно для вибору файлу з'являється у режимі виконання додатку при виклику метода **ShowDialog()**. Коли користувач у діалоговому вікні натисне на кнопку ВІДКРИТИ, метод **ShowDialog()** повертає значення **DialogResult**, яке порівнюється зі значенням такої ж властивості у форми. Для форми значення властивості =ОК.

Основні властивості компоненту:

**Title** - Текст у заголовку вікна.

**InitialDirectory** - Каталог, вміст якого відображається при появі діалогового вікна.

**FileName** – Ім'я обраного користувачем файлу

**Multiselect** – дозволяє вибрати групу файлів.

**Filter** – умовна фільтрація файлу.

Text files (\*.txt) | \*.txt | All files (\*.\*) | \*.\*

**ShowReadOnly** – дає можливість з'явитися індикатору поряд з файлом, якщо він має тип «тільки для читання»

## Наприклад. Подія, яка викликає діалогове вікно для відкриття файлу

```
private: System::Void відкритиToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)
{
    //Відкрити документ (меню)
    openFileDialog1->Filter = "Jpg files|.jpg|Bmp files|.bmp|All files (*.*)|*.*";
    openFileDialog1->FileName="";
    if( openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
    image = gcnew Bitmap(Image::FromFile(openFileDialog1->FileName), pictureBox1->Width, pictureBox1->Height);
    pictureBox1->Image = image;
    this->Text+=(openFileDialog1->FileName);
}
}
```

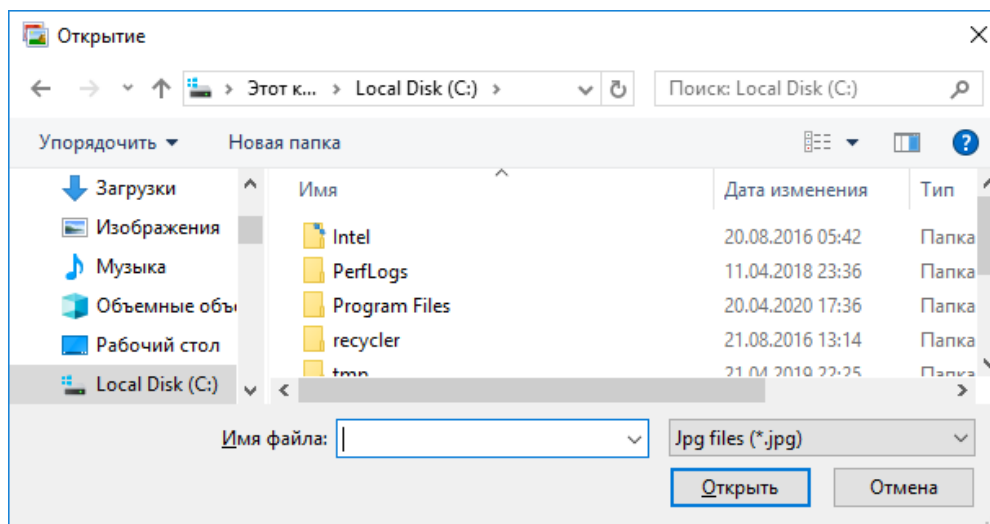


Рис.24. Приклад діалогового вікна

### Компонент *SaveFileDialog*.

Компонент для відображення діалогового вікна «**Сохранить**» та збереження файлу. Але компонент дає лише шлях до місця розташування файлу, написання коду, який відповідає за збереження даних відводиться програмісту.

Діалогове вікно з'являється у режимі виконання додатку при виклику метода **ShowDialog()**. Більшість властивостей компоненту аналогічна компоненту **OpenFileDialog**. Для відображення вікна «Сохранить как» необхідно значення властивості **OverwritePromp = true**.

Коли користувач обирає ім'я файлу та натискає кнопку збереження метод **ShowDialog()** заносить у властивість **FileName** компоненту ім'я файлу та шлях до нього. При цьому жодного перезапису файлу не відбувається.

### *Наприклад. Подія, яка викликає діалогове вікно для збереження файлу*

```
private: System::Void зберегтиToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)
{
    //Зберегти (меню)
    saveFileDialog1->Filter = "bmp files (*.bmp)|*.bmp|jpg files (*.jpg)|*.jpg|All files (*.*)|*.*";
    if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)
        image->Save(saveFileDialog1->FileName);
    this->Text+=(saveFileDialog1->FileName);
}
```

**Компонент ColorDialog** - викликає діалогове вікно вибору кольору. Значення кольору записується у властивість **Color**.

```
private: System::Void toolStripButton6_Click(System::Object^ sender, System::EventArgs^ e)
{
    //палітра кольорів (панель інструментів)
    this->colorDialog1->ShowDialog();
    myPen->Color=this->colorDialog1->Color;
    myBrush->Color=this->colorDialog1->Color;
}
```

**Компонент FontDialog** – виклик вікна налаштування параметрів шрифту. Обрані параметри присвоюються властивості компоненту **Font**.

**Компонент PrintDialog** – відкриває діалогове вікно налаштування параметрів друку

```
private: System::Void toolStripButton4_Click(System::Object^ sender, System::EventArgs^ e)
{
    //Виведення на друк
    if (printDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) printDocument1->Print();
}
```

## ТЕМА 4. ГРАФІЧНІ МОЖЛИВОСТІ МОВИ C++

### Компонент PictureBox

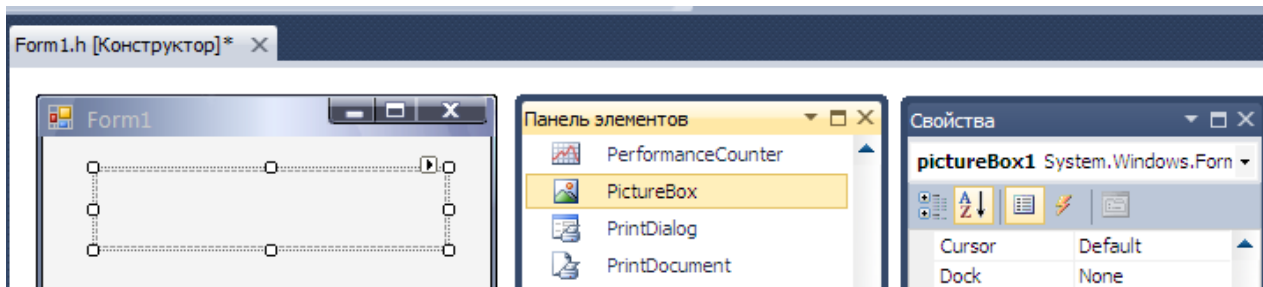


Рис.25. Компонент *PictureBox*

Відображення графіки забезпечує компонент **PictureBox**. Графічна поверхня компоненту **PictureBox** є об'єктом **Graphics**, методи якого і забезпечують виведення графіки. Таким чином, для того, щоб на поверхні компоненту **PictureBox** з'явилася лінія, прямокутник, коло або завантажена з файлу ілюстрація, необхідно викликати відповідний метод об'єкту **Graphics**.

Таблиця 4. Властивості компоненту *PictureBox*

Властивість	Опис властивостей компоненту
<b>Image</b>	Ілюстрація, яка відображається в полі компоненту
<b>SizeMode</b>	Режим відображення ілюстрації (спосіб масштабування), якщо розмір ілюстрації не відповідає розміру компоненту: <b>Normal</b> - масштабування не виконується (якщо розмір компонента менше розміру ілюстрації, то відображається тільки частина ілюстрації); <b>StretchImage</b> - виконується масштабування ілюстрації таким чином, що вона займає всю область відображення (якщо розмір компоненту не пропорційний розміру ілюстрації, вона буде спотворена); <b>AutoSize</b> - розмір компоненту автоматично змінюється і відповідає розміру ілюстрації; <b>CenterImage</b> - центрування ілюстрації в полі компоненту, якщо розмір ілюстрації менше розміру області відображення; <b>Zoom</b> - виконується масштабування так, щоб ілюстрація в полі компоненту відображалася без спотворення

<b>Location</b>	Положення компоненту (області відображення ілюстрації) на поверхні форми. Що уточнює властивість <b>X</b> визначає відстань від лівої межі області до лівої межі форми, що уточнює властивість <b>Y</b> - від верхньої межі області до верхньої межі клієнтської області форми (нижньої межі заголовка)
<b>Dock</b>	Спосіб прив'язки компоненту до компоненту-батька. Компонент може бути прив'язаний до лівої межі ( <b>Left</b> ), правої ( <b>Right</b> ), верхньої ( <b>Top</b> ), нижньої ( <b>Bottom</b> ) або займати всю вільну область компоненту-батька ( <b>Fill</b> )
<b>Size</b>	Розмір компоненту (області відображення ілюстрації). Що уточнює властивість <b>Width</b> визначає ширину області, <b>Height</b> - висоту
<b>Visible</b>	Ознака указує, чи відображається компонент і, відповідно, ілюстрація на поверхні форми
<b>BorderStyle</b>	Вид межі компоненту: <b>None</b> - межа не відображається; <b>FixedSingle</b> - тонка; <b>Fixed3D</b> - об'ємна
<b>Graphics</b>	Поверхня, на яку можна виводити графіку (доступ до графічної поверхні компоненту є у процедури обробки події <b>Paint</b> )

Ілюстрацію, що відображається в полі компоненту **PictureBox**, можна задати під час розробки форми або завантажити з файлу під час роботи програми. Щоб задати ілюстрацію під час створення форми, треба в рядку властивості **Image** клацнути на кнопці з трьома крапками і в стандартному вікні, що з'явилося, «Открыть» вибрати файл ілюстрації. Середовище розробки створить бітовий образ ілюстрації і помістить його у файл ресурсів (таким чином, надалі файл ілюстрації програмі буде не потрібен). Завантаження ілюстрації з файлу під час роботи програми забезпечує метод **FromFile**. Як параметр методу треба вказати ім'я файлу ілюстрації.

### **Наприклад**

```
pictureBox1->Image = System::Drawing::Bitmap::FromFile("d:\Pict.jpg");
```

Дана інструкція забезпечує завантаження і відображення ілюстрації, яка знаходиться у файлі d:\Photo\Pict0025.jpg. Метод **FromFile** дозволяє працювати з файлами BMP, JPEG, GIF, PNG і інших форматів.



## Виведення графічних зображень

Доступ до графічної поверхні об'єкту (властивості **Graphics**) є тільки у функції обробки події **Paint**. Подія **Paint** виникає на початку роботи програми, коли її вікно і всі компоненти з'являються на екрані, а також під час роботи програми всякий раз, коли вікно або його частина знов з'являється на екрані, наприклад.

Для створення процедури обробки події **Paint** необхідно вибрати компонент **PictureBox**, потім у вікні **Properties** відкрити вкладку **Events** і в поле події **Paint** ввести ім'я функції обробки події (*або зробити подвійне клацання лівою кнопкою миші*).

Графічна поверхня складається з окремих точок - пікселів. Положення точки на графічній поверхні характеризується горизонтальною (X) і вертикальною (Y) координатами. Координати точок відраховуються від лівого верхнього кута і зростають зліва направо (координата X) і зверху вниз (координата Y). Ліва верхня точка графічної поверхні має координати (0, 0). Розмір графічної поверхні форми відповідає розміру клієнтської області (*тобто без урахування висоти області заголовка і ширини меж*) форми (властивість **ClientSize**), а розмір графічної поверхні компоненту **PictureBox** - розміру компоненту.

## Олівці і пензлі

Методи малювання графічних примітивів (наприклад, **DrawLine** - лінія, **DrawRectangle** - прямокутник, **FillRectangle** - область) використовують олівці і пензлі. Олівець (об'єкт **Pen**) визначає вид лінії, пензель (об'єкт **Brush**) - вид заливки області.

Метод

```
DrawLine(Pens::Black,10,20,100,20);
```

малює з точки (10, 20) горизонтальну лінію, використовуючи чорний (**Black**) олівець із стандартного набору олівців (**Pens**).

Метод

```
FillRectangle(Brushes::Green,5,10,20,20);
```

за допомогою стандартного пензля зеленого (**Green**) кольору малює зелений квадрат, лівий верхній кут якого знаходиться в точці (5, 10).

Олівці і пензлі визначені в просторі імен **System::Drawing**.

### Олівець

Олівець визначає вид лінії - **колір, товщину і стиль**. У розпорядженні програміста є два набори олівців: **стандартний і системний**. Також програміст може створити власний олівець.

**Стандартний набір олівців** - це кольорові олівці (всього їх 141), які малюють безперервну лінію товщиною в *один піксель*.

Таблиця 5. Кольорова гамма олівців

Олівці	Колір
Pens::Red	Червоний
Pens::Orange	Помаранчевий
Pens::Yellow	Жовтий
Pens::Green	Зелений
Pens::LightBlue	Блакитний
Pens::Blue	Синій
Pens::Purple	Пурпуровий
Pens::Black	Чорний
Pens::LightGray	Сірий
Pens::White	Білий
Pens::Transparent	Прозорий

**Системний набір олівців** - це олівці, колір яких визначається поточною колірною схемою операційної системи і співпадає з кольором якого-небудь елемента інтерфейсу користувача. Наприклад, колір олівця **SystemPens::ControlText** співпадає з кольором, який в поточній колірній схемі використовується для відображення тексту на елементах управління (командних кнопках і ін.), а колір олівця **SystemPens::WindowText** - з кольором тексту у вікнах повідомлень.

Олівець із **стандартного (Pens)** і **системного (SystemPens)** наборів малює безперервну лінію товщиною в один піксель. Якщо треба намалювати пунктирну лінію або лінію завтовшки більше одиниці, то необхідно використовувати **олівець програміста**.

**Олівець програміста** - це об'єкт типу **Pen**, властивості якого визначають вид лінії, що малюється олівцем. Колір, ширину лінії і стиль олівця, створеного програмістом, можна змінити. Щоб це зробити, треба встановити значення відповідної властивості.

```
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
System::Drawing::Pen^ aPen; // олівець
// створити червоний «товстий» олівець
aPen = gnew System::Drawing::Pen(Color::Red,2);
e->Graphics->DrawRectangle(aPen,10,10,100,100);
// створити зелений олівець товщиною 4 пікселя
aPen->Width = 4;
aPen->Color = Color::Green;
....
// стиль лінії - пунктирний
aPen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dash;
....
}
```

### **Пензель**

Пензлі використовуються для зафарбовування внутрішніх областей геометричних фігур.

#### ***Наприклад***

```
e->Graphics->FillRectangle(Brushes::DeepSkyBlue, x, y, w, h);
```

Дана інструкція малює зафарбований прямокутник. **Brushes::DeepSkyBlue** - це стандартний пензель темно-небесного кольору. Параметри **x**, **y** визначають положення прямокутника; **w**, **h** - його розмір.

У розпорядженні програміста є три типу пензлей: **стандартні**, **штрихові** і **текстурні**.

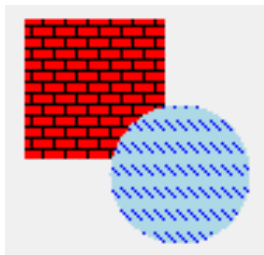
**Стандартний пензель** зафарбовує область одним кольором (суцільне зафарбовування). Всього є 140 пензлів.

Таблиця 6. Кольорова гамма пензлів

Пензлі	Колір
Brushes::Red	Червоний
Brushes::Orange	Помаранчевий
Brushes::Yellow	Жовтий
Brushes::Green	Зелений
Brushes::LightBlue	Блакитний
Brushes::Blue	Синій
Brushes::Purple	Пурпуровий
Brushes::Black	Чорний
Brushes::LightGray	Сірий
Brushes::White	Білий
Brushes::Transparent	Прозорий

**Штриховий пензель (HatchBrush)** зафарбовує область шляхом штрихування. Область може бути заштрихована горизонтальними, вертикальними або лініями під кутом різного стилю і товщини.

### *Приклад застосування штрихового пензля*



```
// необхідно підключити
using namespace System::Drawing::Drawing2D;
так як HatchBrush - це клас простору імен
System.Drawing.Drawing2D

HatchBrush ^myBrush=gcnew
HatchBrush(HatchStyle::HorizontalBrick, Color::Black,
Color::Red);
e->Graphics->FillRectangle(myBrush, 70, 70, 50, 50);

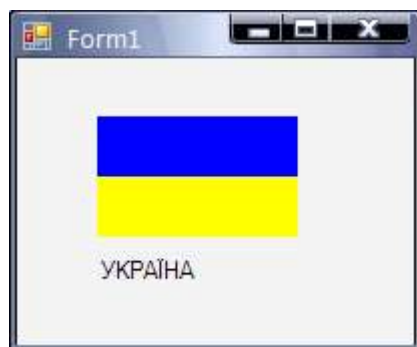
HatchBrush ^fBrush=gcnew
HatchBrush(HatchStyle::DashedDownwardDiagonal, Color::Blue, Color:
:LightBlue);
e->Graphics->FillEllipse(fBrush, 100, 100, 50, 50);
```

**Текстурний пензель (TextureBrush)** є малюнком, який звичайно завантажується під час роботи програми з файлу (bmp, jpg або gif) або з ресурсу.

Зафарбовування області виконується шляхом дублювання малюнка у середині області.

```
TextureBrush^ paintBrush=gcnew
TextureBrush(Image::FromFile("d:p.jpg"));
e->Graphics->FillRectangle(paintBrush, 150,150,150, 150);
```

*Приклад програми «Зображення прапора України».*



```
private: System::Void pictureBox1_Paint(...)
{
    int x =10, y=10; // лівий верхній кут
    int w =100, h=30; // ширина та висота смуг
    e->Graphics->FillRectangle(Brushes::Blue,x,y,w,h); // блакитна смуга
    y += h;
    e->Graphics->FillRectangle(Brushes::Yellow,x,y,w,h); // жовта смуга
    // Підпис. Використовується шрифт, заданий властивістю Font форми.
    e->Graphics->DrawString("УКРАЇНА",this->Font, Brushes::Black, 10, 80);
}
```

## Графічні примітиви

Будь-яка картинка, креслення, схема є сукупність графічних примітивів: крапок, ліній, кіл, дуг, тексту і ін. Побудова графічних примітивів на графічній поверхні (**Graphics**) виконують відповідні методи.

*Таблиця 7. Методи побудови графічних примітивів*

Метод	Дія
<b>DrawLine(Pen, x1, y1, x2, y2)</b>	Лінія. Параметр <b>Pen</b> визначає колір, товщину і стиль лінії; параметри x1, y1, x2, y2 або p1 і p2 - координати точок почала і кінця лінії
<b>DrawRectangle(Pen, x, y, w, h)</b>	Прямокутник. Параметр <b>Pen</b> визначає колір, товщину і стиль межі прямокутника: параметри x, y - координати лівого верхнього кута; параметри w і h задають розмір прямокутника

<b>FillRectangle(Brush, x, y, w, h)</b>	Малює зафарбований прямокутник. Параметр <b>Brush</b> визначає колір і стиль зафарбовування прямокутника; параметри <b>x, y</b> - координати лівого верхнього кута; параметри <b>w i h</b> задають розмір прямокутника
<b>DrawEllipse(Pen, x, y, w, h)</b>	Малює еліпс (контур). Параметр <b>Pen</b> визначає колір, товщину і стиль лінії еліпса; параметри <b>x, y, w, h</b> - координати лівого верхнього кута і розмір прямокутника, усередині якого викреслюється еліпс
<b>FillEllipse(Brush, x, y, w, h)</b>	Малює закрашений еліпс. Параметр <b>Brush</b> визначає колір і стиль зафарбовування внутрішньої області еліпса; параметри <b>x, y, w, h</b> - координати лівого верхнього кута і розмір прямокутника, усередині якого викреслюється еліпс
<b>DrawPolygon(Pen, P)</b>	Малює контур багатокутника. Параметр <b>Pen</b> визначає колір, товщину і стиль лінії межі багатокутника; параметр <b>P</b> (масив типу <b>Point</b> ) - координати кутів багатокутника
<b>FillPolygon(Brush, P)</b>	Малює закрашений багатокутник. Параметр <b>Brush</b> визначає колір і стиль зафарбовування внутрішньої області багатокутника; параметр <b>P</b> (масив типу <b>Point</b> ) - координати кутів багатокутника
<b>DrawString(str, Font, Brush,x,y)</b>	Виводить на графічну поверхню рядок тексту. Параметр <b>Font</b> визначає шрифт; <b>Brush</b> - колір символів; <b>x i y</b> - точку, від якої буде виведений текст
<b>DrawImage(Image, x, y)</b>	Виводить на графічну поверхню ілюстрацію. Параметр <b>Image</b> визначає ілюстрацію; <b>x i y</b> - координату лівого верхнього кута області виведення ілюстрації

Один і той же елемент можна намалювати за допомогою різних, але методів, що мають однакові імена (пригадайте: можливість оголошення функцій, що мають однакові імена, але різні параметри, називається перевантаженням).

Наприклад, прямокутник можна намалювати методом **DrawRectangle**, якому як параметри передаються координати лівого верхнього кута і розміри прямокутника:

```
e->Graphics->DrawRectangle(Pens::Black, x, x, w, h)
```

Це ж завдання може вирішити метод **DrawRectangle**, якому як параметр передається структура типу **Rectangle**, поля якої визначають прямокутник (положення і розмір):

```
Rectangle aRect = Rectangle(20,100,50,50);
```

```
e->Graphics->DrawRectangle(Pens::Blue, aRect);
```

Існування декількох методів, що виконують одне і те ж завдання, дозволяє програмісту вибрати метод, найбільш відповідний для вирішення конкретного завдання.

Як параметри методів викреслювання графічних примітивів часто використовується структура **Point**. Її поля **X** і **Y** визначають положення (координати) точки графічної поверхні.

*Наприклад:*

```
Point p1 = Point(10,10);
```

```
Point p2 = Point(100,10);
```

```
e->Graphics->DrawLine(Pens::Green, p1, p2); // малюємо лінію з p1 у p2
```

### **Методи побудови графічних примітивів**

Прямокутник можна намалювати

1. Метод **DrawRectangle**, якому як параметри передаються координати лівого верхнього кута і розміри прямокутника:

```
e->Graphics->DrawRectangle(Pens::Black, x, y, w, h)
```

2. Метод **DrawRectangle**, якому як параметр передається структура типу **Rectangle**, поля якої визначають прямокутник (положення і розмір):

```
Rectangle aRect = Rectangle(20,100,50,50);
```

```
e->Graphics->DrawRectangle(Pens::Blue, aRect);
```

3. Структура **Point**. Її поля **X** і **Y** визначають положення (координати) точки графічної поверхні.

```
Point p1 = Point(10,10);
```

```
Point p2 = Point(100,10);
```

```
// малюємо лінію з p1 у p2
```

```
e->Graphics->DrawLine(Pens::Green, p1, p2);
```

## **Лінія**

Метод **DrawLine** малює пряму лінію. У інструкції виклику методу слід вказати олівець, яким треба намалювати лінію, і координати точок почала і кінця лінії:

```
DrawLine(aPen x1, y1, x2, y2)
```

Параметр **aPen** задає олівець, який малює лінію:  $x_1$  і  $y_1$  - точка початку лінії, а  $x_2$  і  $y_2$  - точка кінця лінії. Параметри  $x_1$ ,  $y_1$ ,  $x_2$  і  $y_2$  повинні бути одного типу (Integer або Single).

### **Наприклад:**

```
e->Graphics->DrawLine(Pens::Green,10,10,300,10);
```

Дана інструкція малює зелену лінію товщиною в один піксел з точки (10, 10) у точку з координатами (300, 10).

## **Ламана лінія**

Метод **DrawLines** малює ламану лінію. Як параметри методу передається олівець (**Pen**) і масив типу **Point**, елементи якого містять координати вузлових точок лінії. Метод малює ламану лінію, послідовно сполучаючи крапки, координати яких знаходяться в масиві: першу з другою, другу з третьою, третю з четвертою і т.д.

*Приклад фрагменту коду побудови ламаної лінії, що складається з чотирьох ланок.*

```
Array <Point>^ p; // масив крапок
```

```
p = gsnnew array<Point>(5);
```

```
p[0].X = 10; p[0].Y = 50;
```



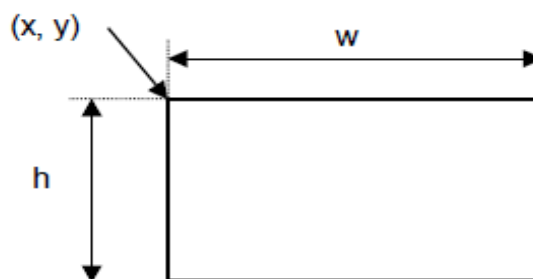
```
p[1].X = 20; p[1].Y = 20;  
p[2].X = 30; p[2].Y = 50;  
p[3].X = 40; p[3].Y = 20;  
p[4].X = 50; p[4].Y = 50;  
e->Graphics->DrawLines(Pens::Green,p);
```

Метод **DrawLines** можна використовувати для побудови замкнутих контурів. Для цього перший і останній елементи масиву повинні містити координати однієї і тієї ж точки.

## **Прямокутник**

Метод **DrawRectangle** креслить прямокутник . Як параметри методу треба вказати олівець, координати лівого верхнього кута і розмір прямокутника:

**DrawRectangle(aPen, x, y, w, h);**



**DrawRectangle(aPen, x, y, w, h)**

Параметри **X** і **Y** задають координати лівого верхнього кута прямокутника, а **W** і **H** - розмір (ширину і висоту).

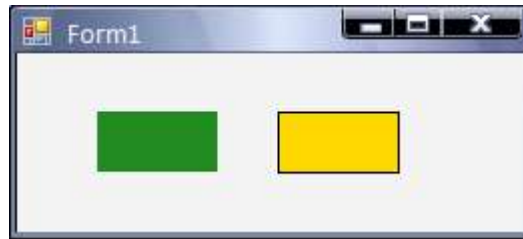
Вид лінії межі прямокутника (*колір, стиль і ширину*) визначає параметр **aPen**, як який можна використовувати один із стандартних олівців або олівець, створений програмістом.

Метод **FillRectangle** малює зафарбований прямокутник. Як параметри методу треба передати пензель, координати лівого верхнього кута і розмір прямокутника:

**FillRectangle(aBrush, x, y, w, h);**

Параметр **aBrush**, як який можна використовувати стандартну або створену програмістом штриховий (**HatchBrush**), градієнтний (**LinearGradientBrush**) або текстурний (**TextureBrush**) пензель, визначає колір і стиль зафарбовування області.

*Приклад програми (процедури обробки події **Paint**), яка демонструє використання методів **DrawRectangle** і **FillRectangle**.*



```
private: System::Void pictureBox1_Paint(...)
{
// Зелений прямокутник розміром 60x30 лівий верхній кут якого в точці (10, 10)
  e->Graphics->FillRectangle(Brushes::ForestGreen, 10,10,60,30);
// Жовтий прямокутник з чорим контуром розміром 60x30 лівий верхній кут якого в
точці (100, 10)
  e->Graphics->FillRectangle(Brushes::Gold, 100,10,60,30); // прямокутник
  e->Graphics->DrawRectangle(Pens::Black, 100,10,60,30); // контур
}
```

### **Точка**

У об'єкту **Graphics** немає методу, який дозволяє намалювати точку а поверхні. Метод **SetPixel** є у об'єкту **Bitmap**. Тому якщо необхідно сформувати картинку з крапок, доведеться створити об'єкт **Bitmap**, сформувати на його поверхні зображення, а потім це зображення за допомогою методу **DrawImage** вивести на графічну поверхню. Але можна поступити простіше - замість крапки вивести квадрат розміром в один піксель.

### **Наприклад:**

```
e->Graphics->FillRectangle(Brushes::Red,x,y,1,1);
```

Дана інструкція малює на графічній поверхні червону крапку.

### **Багатокутник**

Метод **DrawPolygon** креслить багатокутник (контур). Інструкція виклику методу в загальному вигляді виглядає так:

**DrawPolygon(aPen, p)**

Параметр **p** - масив типу **Point**, визначає координати вершин багатокутника. Метод **DrawPolygon** креслить багатокутник, сполучаючи прямими лініями крапки, координати яких знаходяться в масиві: першу з другою, другу з третьою і т.д.

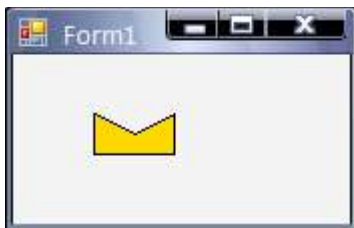
Остання крапка з'єднується з першою. Вид межі багатокутника визначає параметр **aPen**, як який можна використовувати стандартний або створений програмістом олівець.

Зафарбований багатокутник малює метод **FillPolygon**. Інструкція виклику методу в загальному вигляді виглядає так:

### **FillPolygon(aBrush, p)**

Параметр **aBrush**, як який можна використовувати стандартну або створену програмістом штрихову (**HatchBrush**), градієнтну (**LinearGradientBrush**) або текстуровану (**TextureBrush**) кисть, визначає колір і стиль зафарбовування внутрішньої області багатокутника.

*Приклад фрагменту коду, який демонструє використання методів **DrawPolygon** і **FillPolygon** - малює зафарбований жовтий багатокутник з чорним контуром.*



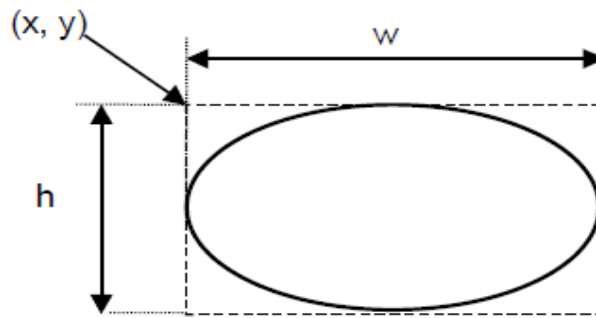
```
array<Point>^ p;  
p = gnew array<Point>(5);  
p[0].X = 10; p[0].Y = 30;  
p[1].X = 10; p[1].Y = 10;  
p[2].X = 30; p[2].Y = 20;  
p[3].X = 50; p[3].Y = 10;  
p[4].X = 50; p[4].Y = 30;  
e->Graphics->FillPolygon(Brushes::Gold, p);  
e->Graphics->DrawPolygon(Pens::Black, p);
```

### **Еліпс і коло**

Метод **DrawEllipse** креслить еліпс усередині прямокутної області. Якщо прямокутник є квадратом, то метод малює коло.

Інструкція виклику методу **DrawEllipse** в загальному вигляді:

**DrawEllipse (aPen, x, y, w, h);**



**DrawEllipse(aPen, x, y, w, h)**

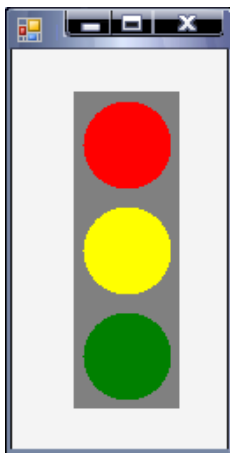
Параметр **aPen**, як який можна використовувати один із стандартних олівців або олівець, створений програмістом, визначає вид (колір, товщину, стиль) межі еліпса. Параметри **x**, **y**, **w** і **h** задають координати лівого верхнього кута і розмір прямокутника, усередині якого метод малює еліпс.

Метод **FillEllipse** малює зафарбований еліпс. У інструкції виклику методу слід вказати пензель (стандартну або створену програмістом), координати і розмір прямокутника, усередині якого треба намалювати еліпс:

**FillEllipse(aBrush, x, y, w, h);**

Пензель визначає колір і спосіб зафарбовування внутрішньої області еліпса.

### *Приклад програми побудови графічного зображення «Світлофор»*



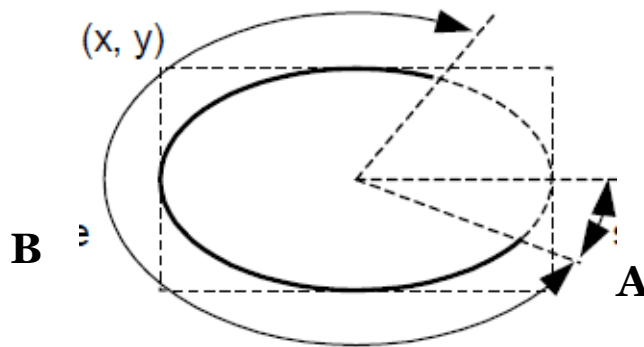
```
private: System::Void pictureBox1_Paint(...)
{
e->Graphics->FillRectangle(Brushes::Gray,5,5,60,180);
e->Graphics->FillEllipse(Brushes::Red, 10,10,50,50);
e->Graphics->FillEllipse(Brushes::Yellow, 10,70,50,50);
e->Graphics->FillEllipse(Brushes::Green, 10,130,50,50);
}
```

## Дуга

Метод **DrawArc** малює дугу - частина еліпса. Інструкція виклику методу в загальному вигляді виглядає так:

**DrawArc(aPen, x, y, w, h, A, B)**

Параметри **x**, **y**, **w** і **h** визначають еліпс (коло), частиною якого є дуга. Параметр **B** задає початкову точку дуги - перетин еліпса і прямої, проведеної з центру еліпса і створюючої кут **A** з горизонтальною віссю еліпса (кутова координата зростає за годинниковою стрілкою). Параметр **B** задає довжину дуги (у градусах). Якщо значення **B** позитивне, то дуга малюється від початкової точки за годинниковою стрілкою, якщо від'ємне - то проти. Величини кутів задаються в градусах.



У інструкції виклику методу **DrawArc** замість параметрів **x**, **y**, **w** і **h** можна вказати структуру типу **Rectangle**:

**DrawArc(aPen, aRect, A, B)**

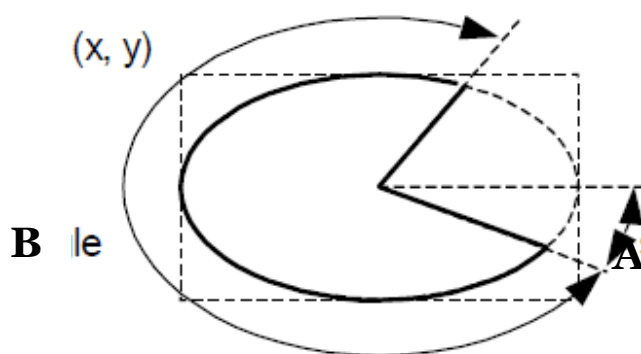
## Сектор

Метод **DrawPie** малює межу сектора. Інструкція виклику методу:

**DrawPie(aPen, x, y, w, h, A, B);**

Параметри **x**, **y**, **w** і **h** визначають еліпс, частиною якого є сектор. Параметр **A** задає початкову точку дуги сектора - перетин еліпса і прямої, проведеної з центру еліпса і створюючої кут **A** з горизонтальною віссю еліпса (кутова координата зростає за годинниковою стрілкою). Параметр **B** - довжину дуги сектора (у градусах). Якщо значення **B** позитивне, то дуга сектора малюється від

початкової точки за годинниковою стрілкою, якщо негативне - проти. Величини кутів задаються в градусах.



У інструкції виклику методу **DrawPie** замість параметрів **x**, **y**, **w** і **h** можна вказати структуру типу **Rectangle**:

**DrawPie(aPen, aRect, A, B)**

Метод **FillPie** малює сектор. Параметри у методу **FillPie**, за винятком першого, замість якого треба вказати кисть, такі ж, як і у методу **DrawPie**.

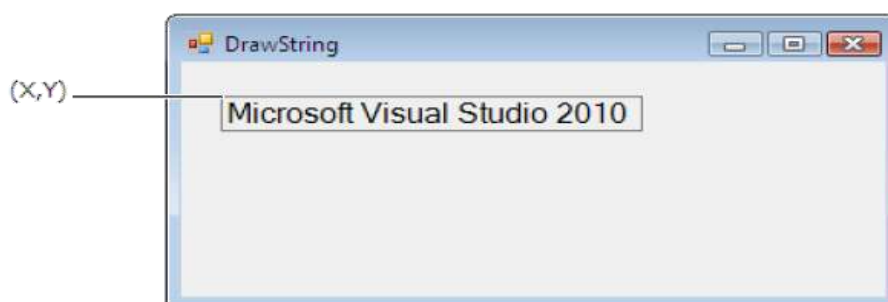
### **Текст**

Виведення тексту на графічну поверхню виконує метод **DrawString**. У інструкції виклику методу вказується рядок, шрифт, кисть і координати точки, від якої треба вивести текст:

**DrawString(st, aFont, aBrush, x, y);**

Параметр **st** задає текст, параметр **aFont** - шрифт, який використовується для відображення тексту, а **aBrush** - колір тексту. Параметри **x** і **y** визначають координати лівого верхнього кута області відображення тексту.

*Приклад програми виведення текстового повідомлення.* У наведеному прикладі для виведення тексту використовується шрифт форми, заданий властивістю **Font**.



```

private: System::Void Form1_Paint(...)
{
String^ st1 = "Microsoft Visual Studio 2010";
e->Graphics->DrawString(st1,this->Font, Brushes::Black, 20,20);
}

```

Інструкція створення шрифту :

```

System::Drawing::Font^ aFont = gcnew System::Drawing::Font(FontFamily,Size,FontStyle);

```

### Приклад



```

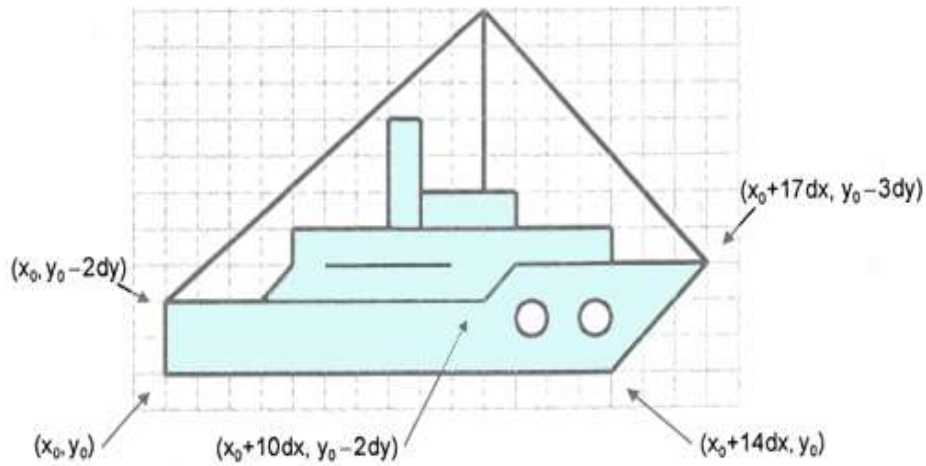
private: System::Void Form1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
int x,y;
x = 10;
y = 10;
String^ st = "Microsoft Visual Studio 2010";
System::Drawing::Font^ rFont = gcnew System::Drawing::Font ("Tahoma", 11,
FontStyle::Regular);
System::Drawing::Font^ bFont = gcnew System::Drawing::Font ("Tahoma", 11,
FontStyle::Bold);
System::Drawing::Font^ iFont = gcnew System::Drawing::Font ("Tahoma", 11,
FontStyle::Italic);
e->Graphics->DrawString(st, rFont, Brushes::Black, x, y);
e->Graphics->DrawString(st, bFont, Brushes::Black, x, y+20);
e->Graphics->DrawString(st, iFont, Brushes::Black, x, y+40);
}

```

### Метод базової точки

При програмуванні складних зображень, що складаються з безлічі елементів, використовується метод, який називається **методом базової точки**. Суть цього методу полягає в наступному:

1. Вибирається деяка точка зображення, яка береться за базову.
2. Координати інших точок відлічуються від базової точки.
3. Якщо координати точок зображення відлічувати від базової у відносних одиницях, а не у пікселях, то забезпечується можливість масштабування зображення.



## Анімація

Під мультиплікацією звичайно розуміється рухомий малюнок. Забезпечити переміщення малюнка досить просто: треба спочатку вивести малюнок на екран, потім через деякий час стерти його і знову вивести цей же малюнок, але вже на деякій відстані від його первинного положення. Підбором часу між виведенням і видаленням малюнка, а також відстані між старим і новим положенням малюнка (крок переміщення), можна досягти того, що у спостерігача складатиметься враження, що малюнок рівномірно рухається по екрану.

Компонент **Timer**, який використовується для генерації послідовності подій, функція обробки яких забезпечує виведення і видалення малюнка. Компонент **Timer** є невізуальним. Це означає, що під час роботи програми компонент в діалоговому вікні не відображається. Тому компонент **Timer** можна помістити в будь-яку точку форми. Властивості компоненту **Timer** приведені у табл. 17.5.

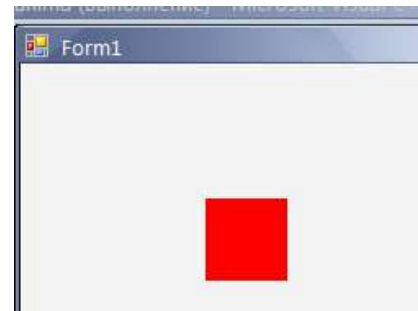
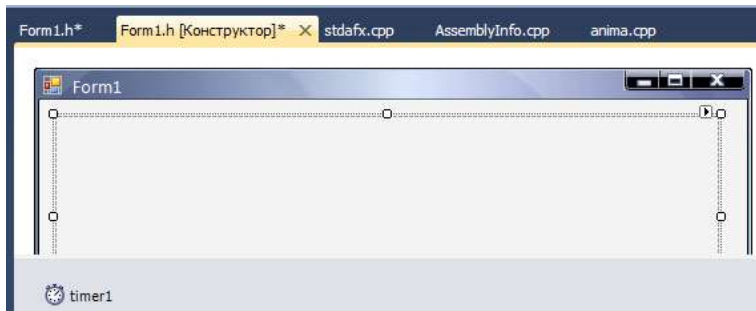
Компонент **Timer** генерує подію **Tick**. Період виникнення події **Tick** вимірюється в мілісекундах і визначається значенням властивості **Interval**. Слід звернути увагу на властивість **Enabled**. Воно дає можливість програмі "запустити" або "зупинити" таймер. Якщо значення властивості **Enabled** = **False**, та подія **Tick** не виникає.

Таблиця 8. Властивості компоненту **Timer**

Властивості	Опис
<b>Interval</b>	Період генерації події <b>Tick</b> . Задається в мілісекундах
<b>Enabled</b>	Дозвіл роботи. Дозволяє (значення <b>True</b> ) або забороняє (значення <b>False</b> ) генерацію події <b>Tick</b>



*Приклад програми, яка імітує рух геометричної фігури (червоний квадрат)*



```
namespace anima {  
...  
using namespace System::Drawing;  
int x=50; // оголошення глобальної змінної  
...  
private: System::Void pictureBox1_Paint (...)  
{ e->Graphics->FillRectangle(Brushes::Red,x,70,50,50); }  
  
private: System::Void timer1_Tick(...)  
{  
    x+=3;  
    pictureBox1->Refresh();  
}
```

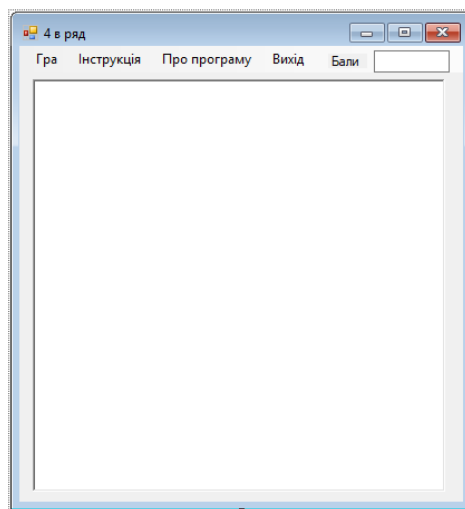
## Етапи розробки комп'ютерної гри

**Формулювання задачі:** На полі фіксованого розміру (наприклад 10x10) розміщено випадковим чином фішки 6 кольорів. Гравець може вказати дві клітинки, які він бажає переставити місцями. Якщо після такої перестановки отримується принаймні одна зв'язна область, яка містить не менше 4 фішок однакового кольору, то всі фішки цієї області знищуються, на їх місце стають фішки згори, які в свою чергу замінюються випадковими фішками. При цьому користувач отримує певну кількість балів. Якщо зв'язні області містять менше 4 фішок, то фішки повертаються на свої місця.

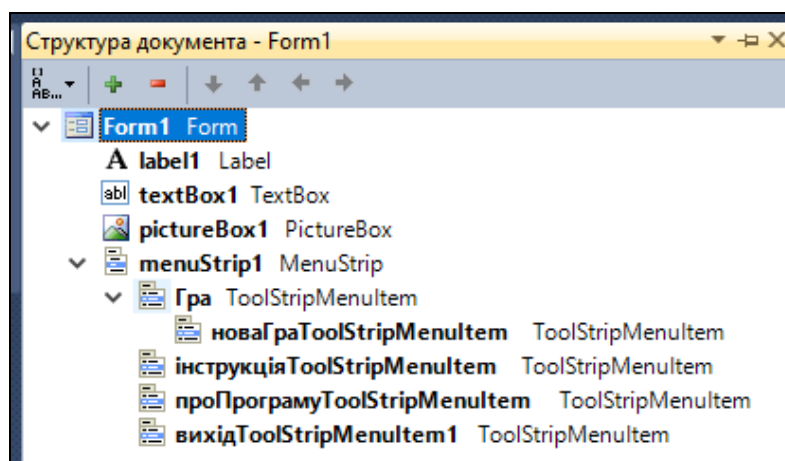
*Високий рівень:* область утворюють фішки одного кольору, які мають сусідів такого ж кольору по горизонталі або по вертикалі, приклад такої області наведено нижче.

		1	
1	1	1	
1		1	1
			1

Для полегшення роботи зі складними задачами процес підготовки завдання для розв'язання на комп'ютері можна розділити на два етапи: створення алгоритму (вимоги до початкових даних та результату, постановка завдання, опис точної схеми розв'язання з вказівкою всіх особливих ситуацій) і написання програми.

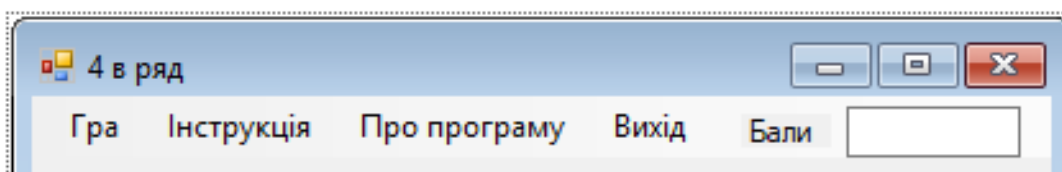


**Перший етап.** Розробка структури документа та елементів графічного інтерфейсу.



До основних компонентів налаштування графічного інтерфейсу користувача середовища Microsoft Visual C++ відносяться:

**Компонент MenuStrip** є головним меню програми. Після того, як у форму буде доданий компонент **MenuStrip**, у верхній частині форми з'являється рядок меню, на початку якого знаходиться область введення тексту. Після того, як структура меню буде сформована, можна виконати налаштування меню шляхом зміни значень властивостей пунктів меню (об'єктів типу ToolStripMenuItem).



При використанні візуального середовища з'являється можливість проектувати деяку частину, наприклад інтерфейси майбутнього продукту, з використанням візуальних засобів додавання і налаштування спеціальних бібліотечних компонентів. Результатом візуального проектування є заготовка майбутньої програми, в яку вже внесені відповідні коди.

У бібліотеці візуальних компонентів існує множина компонентів, що дозволяють відображати, вводити та редагувати текстову інформацію. До основних компонентів використаних у додатку відносяться:

Компонент **Label** призначений лише для відображення текстової інформації (редагування тексту не можливо). Задати текст, що відображається в полі компоненту, можна як під час розробки форми, так і під час роботи програми,

присвоївши значення властивості **Text**. Компонент може бути використаний також для виведення зображень та ідентифікації об'єктів.

Компонент **TextBox** призначений для введення даних з клавіатури. Залежно від налаштування компонента, в полі редагування можна вводити один або декілька (**Multiline = True**) рядків тексту. Всі дані ведені у текстове поле сприймаються програмою, як набір символів. Для перетворення символів до дійсного типу використовують методи **ToDouble** або **ToInt32**.

Відображення графіки забезпечує компонент **PictureBox**. Графічна поверхня компоненту **PictureBox** є об'єктом **Graphics**, методи якого і забезпечують виведення графіки. Таким чином, для того, щоб на поверхні компоненту **PictureBox** з'явилася лінія, прямокутник, коло або завантажена з файлу ілюстрація, необхідно викликати відповідний метод об'єкту **Graphics**.

**Другий етап.** Написання програмного коду.

При завантаженні форми будується таблиця (масив) майбутньої гри.

```
private: System::Void Form1_Load( )
{
    r=gcnew Random();
    for (int i=0;i<10;i++)
    for(int j=0;j<10;j++)
    {
        kyl[i][j]=0;
    }
}
```

При виборі пункту головного меню «**Нова гра**» стає активним компонент **pictureBox1** і створюється ігрове поле.

За побудову сітки відповідає метод **DrawLine**. У інструкції виклику методу слід вказати олівець, яким треба намалювати лінію, і координати точок почала і кінця лінії: **DrawLine(aPen x1, y1, x2, y2)**. Параметр **aPen** задає олівець, який малює лінію: **x1** і **y1** - точка початку лінії, а **x2** і **y2** - точка кінця лінії. Параметри **x1**, **y1**, **x2** і **y2** повинні бути одного типа (**Integer** або **Single**).

Намальована сітка заповнюється різнокольоровими кульками положення та колір яких генерується випадковим чином.

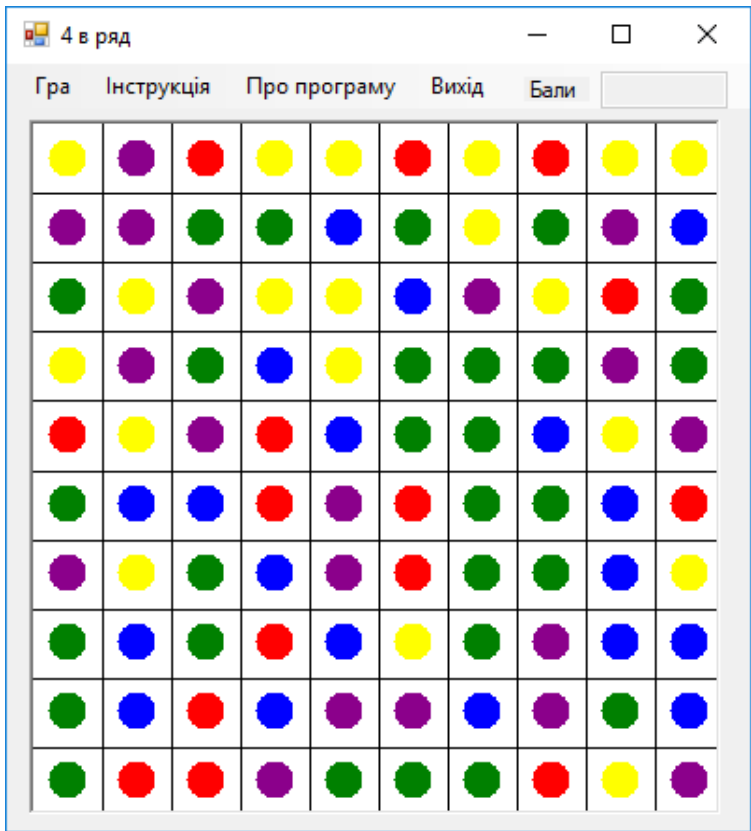
```
{
...
int rand= r->Next(1,6);
kyl[i][j]=rand;
```

```

SolidBrush ^br = gcnw SolidBrush(cl[rand]);
gra->FillEllipse(br,(pictureBox1->Width*i)/10+8,(pictureBox1-
>Width*j)/10+8,20,20);
...
}

```

За виведення зафарбованого кола відповідає метод **FillEllipse**. У інструкції виклику методу слід вказати пензель, координати і розмір прямокутника, усередині якого треба намалювати еліпс **FillEllipse(aBrush, x, y, w, h)**. Пензель **aBrush** визначає колір і спосіб зафарбовування внутрішньої області еліпса. Параметри **x, y, w** і **h** задають координати лівого верхнього кута і розмір прямокутника, усередині якого метод малює еліпс.



За всі операції по вибору фішок відповідає обробник подій **Void pictureBox1\_MouseUp( )**. Дана функція визначає та фіксує положення курсору при відпусканні кнопки миші.

У програмі створені функції: **int funk(int x,int y,int col)** та **funk1()**, які відслідковують процес гри, відповідають за зміни та оновлення поля та повідомляють про перехід на наступний рівень.

Команди головного меню «**Інструкція**», «**Про програму**» (рис.2.3) виводять повідомлення у окремому вікні Тут викликається метод **Show** об'єкту **MessageBox** з

текстом. Оператор розширення області дії (::) вказує системі знайти метод **Show** серед методів об'єкту **MessageBox**.

```
private: System::Void інструкціяToolStripMenuItem_Click( )
{
    MessageBox::Show("Для виконання ходу перемістіть дві фішки так, щоб
    утворилась горизонтальна або вертикальна послідовність з 4 або більше фішок
    одного кольору");
}
```

Отже представлена гра є варіацією класичної гри, коли необхідно зібрати чотири фішки, раніше чим це зробить супротивник (комп'ютер або реальний гравець). Представлений додаток розраховано на одного гравця. Збирати фішки можна по горизонталі чи вертикалі, або групою. Гра вимагає уважності і призначена для користувачів, яким подобаються ігри на логіку.

```
#pragma once
namespace завдання4 {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    int
status=0,CursorX,CursorY,CursorX1,CursorY1,z,z1,z2,z3,perev=0,hodi=0,zarisovka[10][10],n=1;
    int kyl[10][10] = {
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0}
    };
    /// <summary>
    /// Сводка для Form1
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: додайте код конструктора
            //
        }
    protected:
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        ~Form1()
        {
            if (components)
```

```

        {
            delete components;
        }
    }

private: System::Windows::Forms::MenuStrip^ menuStrip1;
private: System::Windows::Forms::ToolStripMenuItem^ Гпа;
protected:

private: System::Windows::Forms::PictureBox^ pictureBox1;
private: System::Windows::Forms::ToolStripMenuItem^ новаГпаToolStripMenuItem;

private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::ToolStripMenuItem^ інструкціяToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^ проПрограмуToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^ вихідToolStripMenuItem1;

private:
    /// <summary>
    /// Требуется переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Обязательный метод для поддержки конструктора - не изменяйте
    /// содержимое данного метода при помощи редактора кода.
    /// </summary>
void InitializeComponent(void)
{
    this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
    this->Гпа = (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->новаГпаToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->інструкціяToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->проПрограмуToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->вихідToolStripMenuItem1 = (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
    this->textBox1 = (gcnew System::Windows::Forms::TextBox());
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->menuStrip1->SuspendLayout();
    (cli::safe_cast<System::ComponentModel::ISupportInitialize^ >(this->pictureBox1))-
>BeginInit();
    this->SuspendLayout();
    //
    // menuStrip1
    //
    this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^
>(4) {this->Гпа, this->інструкціяToolStripMenuItem,
        this->проПрограмуToolStripMenuItem, this->вихідToolStripMenuItem1});
    this->menuStrip1->Location = System::Drawing::Point(0, 0);
    this->menuStrip1->Name = L"menuStrip1";
    this->menuStrip1->Size = System::Drawing::Size(398, 24);
    this->menuStrip1->TabIndex = 0;
    this->menuStrip1->Text = L"menuStrip1";
    // Гпа
    this->Гпа->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^ >(1) {this->новаГпаToolStripMenuItem});
    this->Гпа->Name = L"Гпа";
    this->Гпа->Size = System::Drawing::Size(38, 20);
    this->Гпа->Text = L"Гпа";
    // новаГпаToolStripMenuItem
    this->новаГпаToolStripMenuItem->Name = L"новаГпаToolStripMenuItem";
    this->новаГпаToolStripMenuItem->Size = System::Drawing::Size(152, 22);
    this->новаГпаToolStripMenuItem->Text = L"Нова гпа";
}

```

```

    this->новаГраToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::новаГраToolStripMenuItem_Click);
    //
    // інструкціяToolStripMenuItem
    //
    this->інструкціяToolStripMenuItem->Name = L"інструкціяToolStripMenuItem";
    this->інструкціяToolStripMenuItem->Size = System::Drawing::Size(75, 20);
    this->інструкціяToolStripMenuItem->Text = L"Інструкція";
    this->інструкціяToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::інструкціяToolStripMenuItem_Click);
    //
    // проПрограмуToolStripMenuItem
    //
    this->проПрограмуToolStripMenuItem->Name = L"проПрограмуToolStripMenuItem";
    this->проПрограмуToolStripMenuItem->Size = System::Drawing::Size(99, 20);
    this->проПрограмуToolStripMenuItem->Text = L"Про програму";
    this->проПрограмуToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::проПрограмуToolStripMenuItem_Click);
    //
    // вихідToolStripMenuItem1
    //
    this->вихідToolStripMenuItem1->Name = L"вихідToolStripMenuItem1";
    this->вихідToolStripMenuItem1->Size = System::Drawing::Size(47, 20);
    this->вихідToolStripMenuItem1->Text = L"Вихід";
    this->вихідToolStripMenuItem1->Click += gcnew System::EventHandler(this,
&Form1::вихідToolStripMenuItem1_Click);
    //
    // pictureBox1
    //
    this->pictureBox1->BackColor = System::Drawing::Color::White;
    this->pictureBox1->BorderStyle = System::Windows::Forms::BorderStyle::Fixed3D;
    this->pictureBox1->Cursor = System::Windows::Forms::Cursors::Hand;
    this->pictureBox1->Enabled = false;
    this->pictureBox1->Location = System::Drawing::Point(12, 30);
    this->pictureBox1->Name = L"pictureBox1";
    this->pictureBox1->Size = System::Drawing::Size(370, 372);
    this->pictureBox1->TabIndex = 1;
    this->pictureBox1->TabStop = false;
    this->pictureBox1->Click += gcnew System::EventHandler(this, &Form1::pictureBox1_Click);
    this->pictureBox1->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(this,
&Form1::pictureBox1_MouseUp);
    //
    // textBox1
    //
    this->textBox1->Enabled = false;
    this->textBox1->Location = System::Drawing::Point(318, 4);
    this->textBox1->Name = L"textBox1";
    this->textBox1->Size = System::Drawing::Size(68, 20);
    this->textBox1->TabIndex = 2;
    //
    // label1
    //
    this->label1->AutoSize = true;
    this->label1->Location = System::Drawing::Point(277, 7);
    this->label1->Name = L"label1";
    this->label1->Size = System::Drawing::Size(35, 13);
    this->label1->TabIndex = 3;
    this->label1->Text = L"Бали ";
    //
    // Form1
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
    this->AutoSize = true;
    this->ClientSize = System::Drawing::Size(398, 411);
    this->Controls->Add(this->label1);
    this->Controls->Add(this->textBox1);

```



```

this->Controls->Add(this->pictureBox1);
this->Controls->Add(this->menuStrip1);
this->MainMenuStrip = this->menuStrip1;
this->Name = L"Form1";
this->Text = L"4 в ряд";
this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
this->menuStrip1->ResumeLayout(false);
this->menuStrip1->PerformLayout();
(cli::safe_cast<System::ComponentModel::ISupportInitialize^ >(this->pictureBox1))-
>EndInit();
this->ResumeLayout(false);
this->PerformLayout();

#pragma endregion
public: Random^r;
public: Graphics ^sharik;
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) //створення
таблички в загрузці форми
{
    r=gcnew Random();
    for (int i=0;i<10;i++)
        for(int j=0;j<10;j++)
            {
                kyl[i][j]=0;
            }
}
private: System::Void вихідToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^
e) { }
private: System::Void pictureBox1_MouseUp(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) //обробка подій на відпусканні клавіші мишки
{
    int k=0;
    array<System::Drawing::Color>^c1 = {Color::White, Color::Red, Color::Blue,
Color::DarkMagenta, Color::Green, Color::Yellow, Color::Chocolate};
    if(status==0)
    {
        CursorX = e->X;
        CursorY = e->Y;
        z=CursorX/((pictureBox1->Width)/10);
        z1=CursorY/((pictureBox1->Height)/10);
        status=1;
    }
    else if(status==1)
    {
        CursorX1 = e->X;
        CursorY1 = e->Y;
        z2=CursorX1/((pictureBox1->Width)/10);
        z3=CursorY1/((pictureBox1->Height)/10);
        k++;
        status=0;
        if(((z1==z3)&&(z==z2-1))||((z1==z3)&&(z==z2+1))||((z==z2)&&(z1==z3-
1))||((z==z2)&&(z1==z3+1)))
        {
            int res=0;
            int temp=kyl[z2][z3];
            kyl[z2][z3]=kyl[z][z1];
            kyl[z][z1]=temp;

            Graphics ^gr=pictureBox1->CreateGraphics();
            SolidBrush ^br = gcnew SolidBrush(c1[kyl[z][z1]]);
            gr->FillEllipse(br, (pictureBox1->Width*z)/10+8, (pictureBox1->Width*z)/10+8, 20, 20);

            SolidBrush ^b = gcnew SolidBrush(c1[kyl[z2][z3]]);
            gr->FillEllipse(b, (pictureBox1->Width*z2)/10+8, (pictureBox1->Width*z3)/10+8, 20, 20);

```

```

        res=funk(z,z1,kyl[z][z1]);
        if(res<4)
        {
            res=funk(z2,z3,kyl[z2][z3]);
        }
        else
        {
            funk(z2,z3,kyl[z2][z3]);
        }
        if(res<4)
        {
            int temp=kyl[z2][z3];
            kyl[z2][z3]=kyl[z][z1];
            kyl[z][z1]=temp;

            Graphics ^gr=pictureBox1->CreateGraphics();
            SolidBrush ^br = gcnew SolidBrush(c1[kyl[z][z1]]);
            gr->FillEllipse(br,(pictureBox1->Width*z)/10+8,(pictureBox1->Width*z1)/10+8,20,20);

            SolidBrush ^b = gcnew SolidBrush(c1[kyl[z2][z3]]);
            gr->FillEllipse(b,(pictureBox1->Width*z2)/10+8,(pictureBox1->Width*z3)/10+8,20,20);
        }
        if(n==10)
        {
            pictureBox1->Refresh();
            textBox1->Visible=false;
            pictureBox1->Enabled=false;
            MessageBox::Show("Кінець гри \n"+" "+"Ви набрали"+"
+\"+\"+\" балів за гру \n"+"\"+\"Повертайтеся грати ще!\" ");
            n=1;
        }
    }
    else
    {
        MessageBox::Show("Оберіть інші фішки");
    }
}

private: System::Void новаГраToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) //створення ігрового поля
{
    pictureBox1->Enabled=true;
    textBox1->Visible=true;
    textBox1->Clear();
    array<System::Drawing::Color>^c1 = {Color::White, Color::Red, Color::Blue, Color::DarkMagenta, Color::Green, Color::Yellow, Color::Black};
    Graphics ^g=pictureBox1->CreateGraphics();
    Graphics ^gr=pictureBox1->CreateGraphics();
    Graphics ^gra=pictureBox1->CreateGraphics();
    for(int p=0;p<10;p++)
    {
        g->DrawLine(Pens::Black,(pictureBox1->Width*p)/10,0,(pictureBox1->Width*p)/10,(pictureBox1->Height*p));
    }
    for(int p=0;p<10;p++)
    {
        gr->DrawLine(Pens::Black,0,(pictureBox1->Height*p)/10,(pictureBox1->Width*p),(pictureBox1->Height*p)/10);
    }
    for(int p=0;p<10;p++)
    {
        for(int j=0;j<10;j++)
        {
            int rand= r->Next(1,6);
            SolidBrush ^br = gcnew SolidBrush(c1[rand]);

```

```

        gra->FillEllipse(br, (pictureBox1-
>Width*p)/10+8, (pictureBox1->Width*j)/10+8, 20, 20);
        kyl[p][j]=rand;
    }
}
private: System::Void pictureBox1_Click(System::Object^ sender, System::EventArgs^ e) { }
int funk(int x,int y,int col)
{
    int step=0;
    bool poshyk=true;

    int kolir[10][10];

    for(int i=0;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            if(kyl[i][j]==col)
            {
                kolir[i][j]=-1;
            }
            else
            {
                kolir[i][j]=-2;
            }
        }
    }
    kolir[x][y]=0;
    while (poshyk==true)
    {
        for(int i=0;i<10;i++)
        {
            for(int j=0;j<10;j++)
            {
                if(kolir[i][j]==step)
                {
                    if(i-1 >= 0 && kolir[i][j]!=-2 && kolir[i-1][j]==-1)
                    {
                        kolir[i-1][j]=step+1;
                    }
                    if(j-1 >=0 && kolir[i][j]!=-2 && kolir[i][j-1]==-1)
                    {
                        kolir[i][j-1]=step+1;
                    }
                    if(i+1 <10 && kolir[i][j]!=-2 && kolir[i+1][j]==-1)
                    {
                        kolir[i+1][j]=step+1;
                    }
                    if(j+1 <10 && kolir[i][j]!=-2 && kolir[i][j+1]==-1)
                    {
                        kolir[i][j+1]=step+1;
                    }
                }
            }
        }
        step++;
        if(step>100)
        {
            poshyk=false;
        }
    }
    int znach=0;
}

```

```

for(int i=0;i<10;i++)
{
    for(int j=0;j<10;j++)
    {
        if(kolir[i][j]!=-1 && kolir[i][j]!=-2)
        {
            znach++;
        }
    }
}
if(znach>=4)
{
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            if(kolir[i][j]!=-1 && kolir[i][j]!=-2)
            {
                hodi++;
                textBox1->Text=Convert::ToString(hodi);
                Graphics ^gra=pictureBox1->CreateGraphics();
                gra->FillEllipse(Brushes::White,(pictureBox1-
>Width*i)/10+8,(pictureBox1->Width*j)/10+8,20,20);
                kyl[i][j]=0;
                if(hodi<=100)
                {
                    if(hodi%50==0)
                    {
                        n=hodi/50;
                        n++;
                        MessageBox::Show("Ви
досягли"+" "+n+" "+"рівня");
                    }
                }
            }
        }
    }
}
funkt1();
return znach;
}
void funk1()
{
    array<System::Drawing::Color>^c1 = {Color::White, Color::Red, Color::Blue,
Color::Purple, Color::Green, Color::Yellow, Color::Chocolate};
    Graphics ^gra=pictureBox1->CreateGraphics();
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            if(kyl[i][j]==0 && j==0)
            {
                int rand= r->Next(1,6);
                kyl[i][j]=rand;
                SolidBrush ^br = gcnew SolidBrush(c1[rand]);
                gra->FillEllipse(br,(pictureBox1-
>Width*i)/10+8,(pictureBox1->Width*j)/10+8,20,20);
                j++;
            }
            if(kyl[i][j]==0)
            {
                kyl[i][j]=kyl[i][j-1];
                kyl[i][j-1]=0;
                SolidBrush ^br = gcnew SolidBrush(c1[kyl[i][j]]);
                gra->FillEllipse(br,(pictureBox1-
>Width*i)/10+8,(pictureBox1->Width*j)/10+8,20,20);
                j--;
            }
        }
    }
}

```

```

        else
        {
            j++;
        }
    }
}
private: System::Void інструкціяToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e)
{
    MessageBox::Show("Для виконання ходу перемістіть дві фішки так, щоб
утворилась горизонтальна або вертikalauна послідовність з 4 або більше фішок одного кольору.");
}
private: System::Void вихідToolStripMenuItem1_Click(System::Object^ sender, System::EventArgs^
e) {
    this->Close();
}
};
}

```

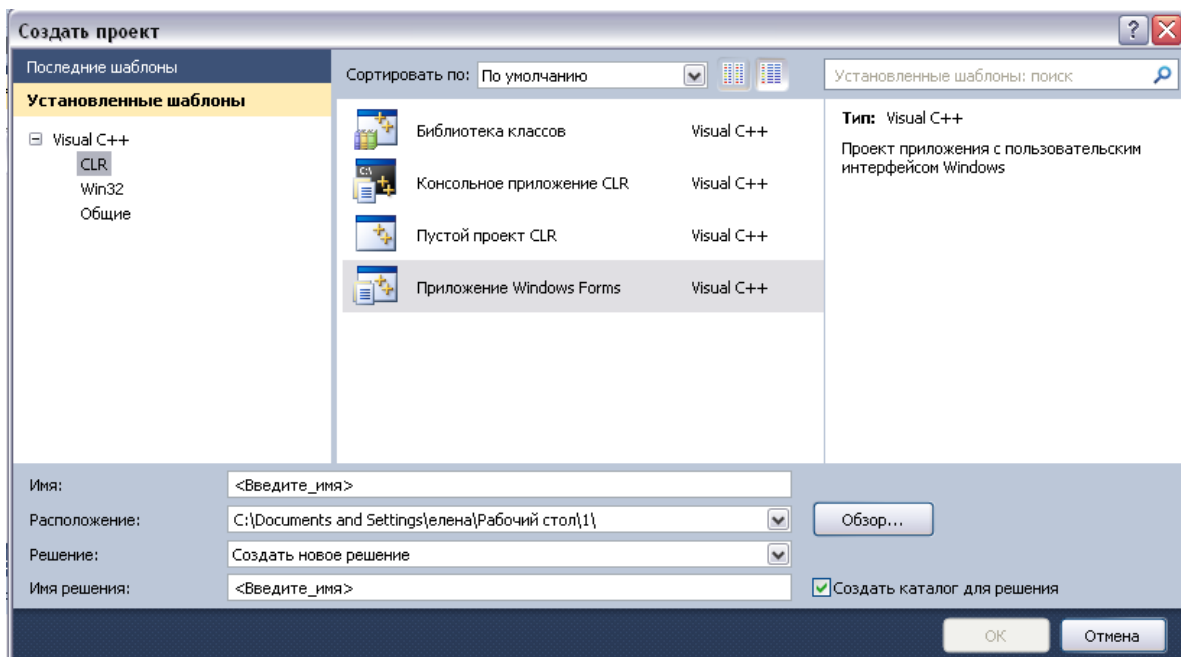
# ЛАБОРАТОРНІ РОБОТИ

## Лабораторна робота №1. Ознайомлення з візуальним середовищем Visual C++. Компіляція та запуск програми на виконання.

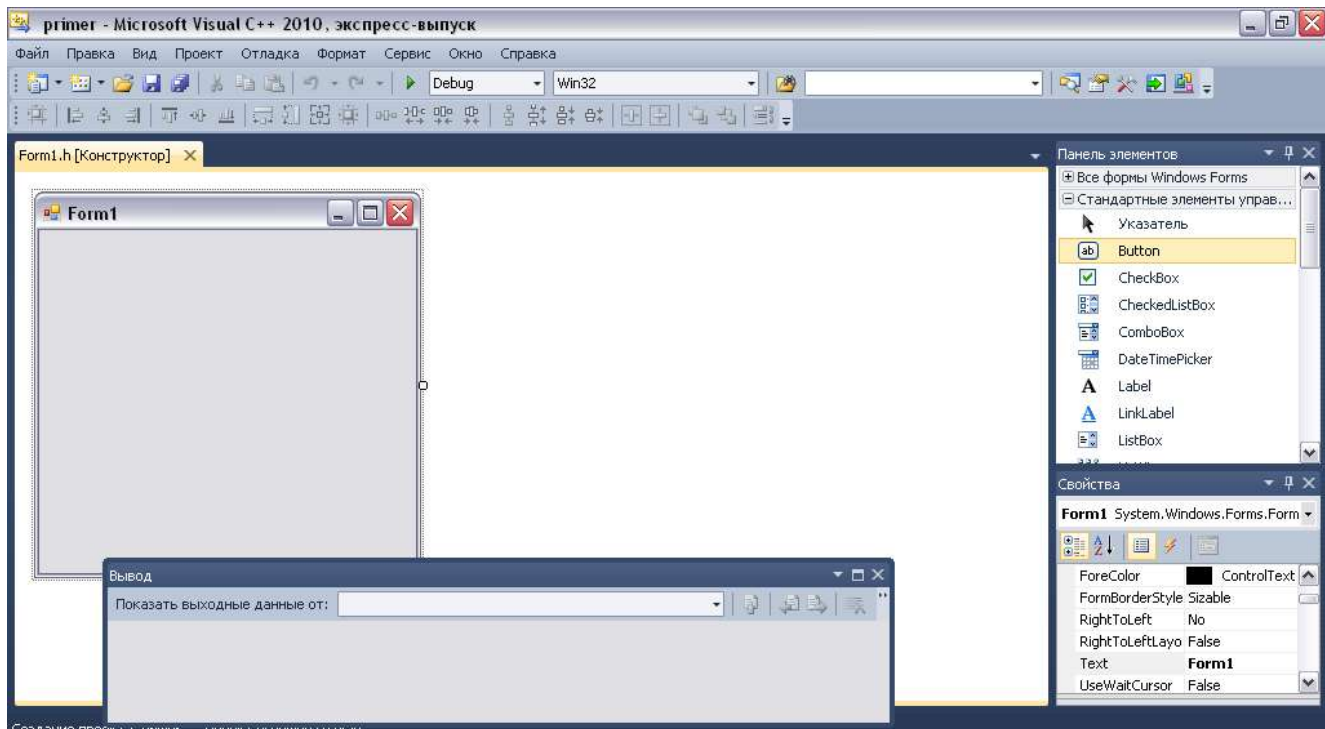
### Теоретичні відомості

1. Після завантаження середовища програмування необхідно виконати команду «Файл-Создать-Проект».

2. У вікні створення проекту Visual C++, області встановлених шаблонів необхідно вибрати **CLR**, а в області шаблонів (у середній колонці) обрати шаблон «**Приложения Windows Forms (Visual C++)**». Після чого необхідно ввести ім'я проекту в поле «Имя» і шлях до файлів проекту.



3. У вікні додатку що створився зображена екранна форма — **Form1**. Форма відобразиться у вкладці **Form1.h [Конструктор]**. У формі розташовують різні компоненти графічного інтерфейсу.



4. Слід пам'ятати, що об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці **Click**.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у властивостях елемента клацнути на зображенні блискавки «События» і в списку всіх можливих подій вибрати подвійним клацанням необхідна подія. Після цього активізується вкладка програмного коду **Form1.h**. Перемикається між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш **Ctrl+Tab**, як це прийнято звичайно при перемиканні між вкладками в Windows.

5. Запуск проекту здійснюється командою «Отладка-Начать отладку» або клавіша **F5** або кнопка із стрілкою на панелі інструментів.

\*\*\*

Для виведення повідомлень у окремому вікні можна використати метод **Show**.

```
MessageBox::Show("Текст");
```

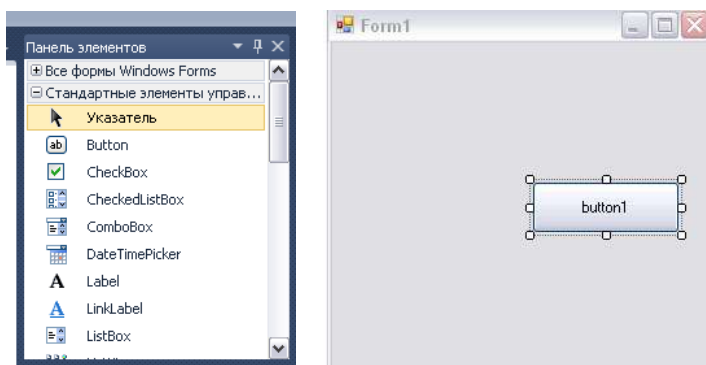
Тут викликається метод **Show** об'єкту **MessageBox** з текстом. Оператор розширення області видимості (::) вказує системі знайти метод **Show** серед методів

об'єкту **MessageBox**. Таким чином, об'єкти окрім властивостей мають також і *методи*, тобто програми, які обробляють об'єкти

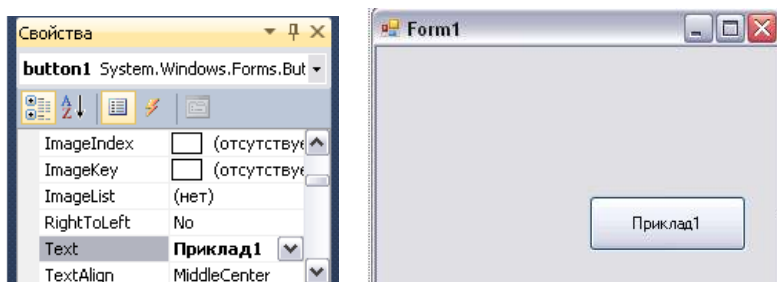
## Завдання для самостійного виконання

### Завдання 1. Виведення повідомлення.

1. Створіть проект. На формі розмістіть кнопку



2. Додайте на кнопку надпис **Приклад1** скориставшись властивістю **Text**.



3. Двічі клікніть по кнопці **Приклад1**. Відкриється вкладка **Form1.h** при цьому сформується порожній обробник подій (**button1\_Click**). Між фігурними дужками {} пропишіть програмний код **MessageBox::Show("Моя перша програма!!!");**

```
Form1.h* x Form1.h [Конструктор]*
primer::Form1
// FORM1
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(292, 266);
this->Controls->Add(this->button1);
this->Name = L"Form1";
this->Text = L"Form1";
this->ResumeLayout(false);
}
#pragma endregion
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    MessageBox::Show("Моя перша програма!!!");
}
```



4. Відкомпілюйте проект та запустіть на виконання «Отладка-Начать отладку».



5. Додайте на форму другу кнопку **ВИХІД** та пропишіть у обробнику подій для елемента **button2** метод **Close( )**. Ви створили кнопку закриття проекту.

**Завдання 2. (відповідно до номеру в списку).** Обчислити значення виразу при заданих початкових даних. Порівняти набутого значення з вказаним правильним результатом.

Результат вивести у окремому вікні (використати метод **Show**).

Додайте на форму кнопку закриття проекту.

**УВАГА!** При використанні математичних функцій необхідно вказувати ім'я класу **Math**. *Наприклад: s=Math::Sqrt(m);*

$$1. s = \frac{2 \cos\left(x - \frac{2}{3}\right)}{\frac{1}{2} + \sin^2 y} \left(1 + \frac{z^2}{3 - z^2/5}\right).$$

При  $x = 14.26$ ;  $y = -1.22$ ;  $z = 3.5 \times 10^{-2}$  відповідь  $s = 0.749155$ .

$$2. s = \frac{\sqrt[3]{9 + (x - y)^2}}{x^2 + y^2 + 2} - e^{|x-y|} \operatorname{tg}^3 z.$$

При  $x = -4.5$ ;  $y = 0.75 \times 10^{-4}$ ;  $z = -0.845 \times 10^2$  відповідь  $s = -3.23765$ .

$$3. s = \frac{1 + \sin^2(x + y)}{\left|x - \frac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(\operatorname{arctg} \frac{1}{z}\right).$$

При  $x = 3.74 \times 10^{-2}$ ;  $y = -0.825$ ;  $z = 0.16 \times 10^2$  відповідь  $s = 1.05534$ .

$$4. s = |\cos x - \cos y|^{(1+2\sin^2 y)} \left( 1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4} \right).$$

При  $x = 0.4 \times 10^4$ ;  $y = -0.875$ ;  $z = -0.475 \times 10^{-3}$  відповідь  $s = 1.98727$ .

$$5. s = \ln \left( y^{-\sqrt{|x|}} \right) \left( x - \frac{y}{2} \right) + \sin^2(\operatorname{arctg}(z)).$$

При  $x = -15.246$ ;  $y = 4.642 \times 10^{-2}$ ;  $z = 21$  відповідь  $s = -182.038$ .

$$6. s = \sqrt{10(\sqrt[3]{x} + x^{y+2})} (\arcsin^2 z - |x - y|).$$

При  $x = 16.55 \times 10^{-3}$ ;  $y = -2.75$ ;  $z = 0.15$  відповідь  $s = -40.6307$ .

$$7. s = 5 \operatorname{arctg}(x) - \frac{1}{4} \arccos(x) \frac{x + 3|x - y| + x^2}{|x - y|z + x^2}.$$

При  $x = 0.1722$ ;  $y = 6.33$ ;  $z = 3.25 \times 10^{-4}$  відповідь  $s = -205.306$ .

$$8. s = \frac{e^{|x-y|} |x-y|^{x+y}}{\operatorname{arctg}(x) + \operatorname{arctg}(z)} + \sqrt[3]{x^6 + \ln^2 y}.$$

При  $x = -2.235 \times 10^{-2}$ ;  $y = 2.23$ ;  $z = 15.221$  відповідь  $s = 39.3741$ .

$$9. s = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right| + (y - x) \frac{\cos y - \frac{z}{(y-x)}}{1 + (y-x)^2}.$$

При  $x = 1.825 \times 10^2$ ;  $y = 18.225$ ;  $z = -3.298 \times 10^{-2}$  відповідь  $s = 1.21308$ .

$$10. s = 2^{-x} \sqrt{x + 4\sqrt{|y|}} \sqrt[3]{e^{x-1/\sin z}}.$$

При  $x = 3.981 \times 10^{-2}$ ;  $y = -1.625 \times 10^3$ ;  $z = 0.512$  відповідь  $s = 1.26185$ .

$$11. s = y^{\sqrt[3]{|x|}} + \cos^3(y) \frac{|x - y| \left( 1 + \frac{\sin^2 z}{\sqrt{x + y}} \right)}{e^{|x-y|} + \frac{x}{2}}.$$

При  $x = 6.251$ ;  $y = 0.827$ ;  $z = 25.001$  відповідь  $s = 0.712122$ .

$$12. s = 2^{(y^x)} + (3^x)^y - \frac{y \left( \operatorname{arctgz} - \frac{1}{3} \right)}{|x| + \frac{1}{y^2 + 1}}.$$

При  $x = 3.251$ ;  $y = 0.325$ ;  $z = 0.466 \times 10^{-4}$  відповідь  $s = 4.23655$ .

$$13. s = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|(\sin^2 z + \operatorname{tg} z)}.$$

При  $x = 17.421$ ;  $y = 10.365 \times 10^{-3}$ ;  $z = 0.828 \times 10^5$  відповідь  $s = 0.330564$ .

$$14. s = \frac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \frac{x + \frac{y}{2}}{2|x+y|} (x+1)^{-1/\sin z}.$$

При  $x = 12.3 \times 10^{-1}$ ;  $y = 15.4$ ;  $z = 0.252 \times 10^3$  відповідь  $s = 82.8256$ .

$$15. s = \frac{x^{y+1} + e^{y-1}}{1+x|y-\operatorname{tg} z|} (1+|y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

При  $x = 2.444$ ;  $y = 0.869 \times 10^{-2}$ ;  $z = -0.13 \times 10^3$  відповідь  $s = -0.498707$ .

## Лабораторна робота №2. Інструменти візуальної розробки додатків.

### Теоретичні відомості

---

Форма є основою додатку. У формі розташовують різні компоненти графічного інтерфейсу користувача.


Це поля для введення тексту **TextBox**, командні кнопки **Button**, рядки тексту у формі — мітки **Label**, які не можуть бути відредаговані користувачем, та інші компоненти. Візуальне програмування, припускає просте перетягування елементів за допомогою миші з **панелі елементів**, де розташовані компоненти користувача, у форму. Це допомагає звести до мінімуму безпосереднє написання програмного коду.

Властивостями кнопки є:

- ім'я кнопки (**Name**) — **button1**;
- напис на кнопці (**Text**);
- розташування кнопки (**Location**) в системі координат форми X, Y;
- розмір кнопки (**Size**);
- активність (**Enabled**);
- видимість (**Visible**);

Властивості можна побачити, якщо клацнути правою кнопкою миші в межах форми і вибрати в контекстному меню команду «Свойства», при цьому з'явиться панель «Свойства» (якщо вона не відображається на екрані).

Слід пам'ятати, що об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці **Click**, клацання в межах форми, завантаження (**Load**) форми в оперативну пам'ять при старті програми.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у панелі «Свойства» клацнути на зображенні блискавки  «События» і в списку всіх можливих подій вибрати подвійним клацанням необхідна подія. Після

цього активізується вкладка програмного коду **Form1.h**. Перемикається між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш **Ctrl+Tab**, як це прийнято звичайно при перемиканні між вкладками в Windows.

Запуск проекту здійснюється командою «**Отладка-Начать отладку**» або клавіша **F5** або кнопка із стрілкою на панелі інструментів.

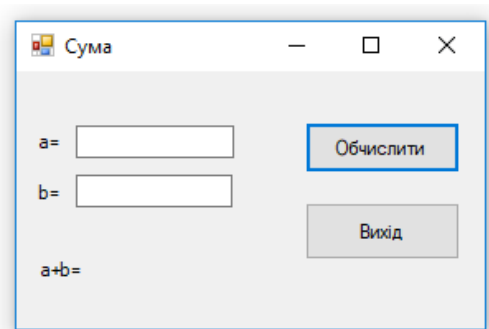
У бібліотеці візуальних компонентів існує множина компонентів, що дозволяють відображати, вводити та редагувати текстову інформацію

## Завдання для самостійного виконання

	1	2	3	4	5	6	7	8	9	10
Завд. 1	1-5	1-5	1-5	1-5	1-5	1-5	1-5	1-5	1-5	1-5
Завд. 2	непарні	парні	непарні	парні	непарні	парні	непарні	парні	непарні	парні
Завд. 3	2, 4	1, 3	4, 2	1, 3	2, 4	1, 3	2, 4	1, 3	2, 4	1, 3

### Завдання 1.

1. Написати програму знаходження суми двох чисел. Напишіть коментарі до кожного рядка програми.



```
private: System::Void button1_Click(...)
{
    double a,s,b;
    a=Convert::ToDouble(textBox1->Text);
    b=Convert::ToDouble(textBox2->Text);
    s=a+b;
    label1->Text=Convert::ToString(s);
}
```

//**Convert::ToDouble** – перетворення змінної до дійсного типу

//**Convert::ToString** - перетворення змінної до рядкового типу

2. Напишіть програму, яка буде за введеним роком вашого народження виводитиме ваш вік.

3. Задано два різних дійсних числа визначте максимальне з них.

4. Написати програму, яка визначає з двох дійсних чисел додатне та збільшує його значення на одиницю (*операція інкременту*).
5. Визначить кількість коренів квадратного рівняння.

### Завдання 2. (Оператор IF)

1. Маємо три дійсних числа  $x, y, z$ . Знайдіть найменше з них.
2. Маємо дійсні додатні числа  $x, y, z$ . Складіть програму, яка буде перевіряти існування трикутника з довжинами сторін  $x, y, z$ . Визначити периметр трикутника.
3. Серед значень функції  $y = \sin x$  у трьох заданих точках  $a, b, c$  визначити кількість різних.
4. Складіть програму, яка перевірятиме введене з клавіатури число на парність.
5. Перевірити чи є введене з клавіатури число додатнім, від'ємним, нулем.
6. Визначити, у якій з трьох даних точок  $a, b, c$  функція  $y = \cos(x)$  набуває найбільшого значення.
7. Дано три різних дійсних числа  $a, b, c$ . Знайти добуток двох менших чисел.
8. Дано три різних дійсних числа  $a, b, c$ . Замінити найменше число добутком двох більших.

### Завдання 3. (Оператор SWITCH)

1. Написати програму, яка за введеним символом математичної операції здійснює відповідну дію над двома числами. При помилковому введенні числа виводиться повідомлення про помилку.

 Методичні вказівки до задачі 1:

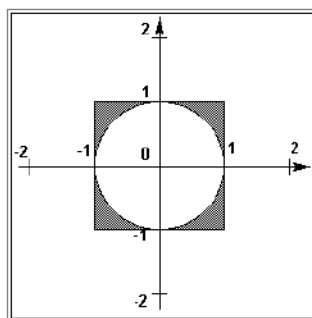
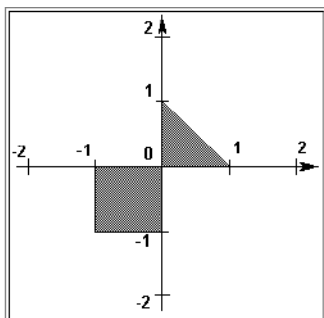
```
...
a=Convert::ToDouble(textBox1->Text);
b=Convert::ToDouble(textBox2->Text);
char c = textBox3->Text[0]; //визначаємо символ операції
switch(c)
{
case '+':s=a+b;break;
...
}
```

2. Написати програму, яка за введеним номером дня тижня виводить його назву. При помилковому введенні числа виводиться повідомлення про помилку.

3. Написати програму переведення оцінки з числового формату в текстовий.( Наприклад 2 – "незадовільно").
4. Написати програму, яка за введеною відповіддю користувача виводить повідомлення про завершення роботи програми. Врахувати регістр літер та мову введення відповіді („у”- продовжити роботу, „н” – завершити.)

### Додаткові завдання

1. Написати програму яка визначає, чи потрапляє точка з координатами  $(x, y)$  у зафарбовану область.



2. Дано три різних дійсних числа. Визначити, яке з них найбільше, найменше та середнє.
3. Відомі рік, номер місяця та день народження двох людей. Визначити хто з них найстарший.
4. Дано три дійсні додатні числа. Якщо існує трикутник з такими сторонами, то визначити його вид (рівносторонній, рівнобедрений, різносторонній)

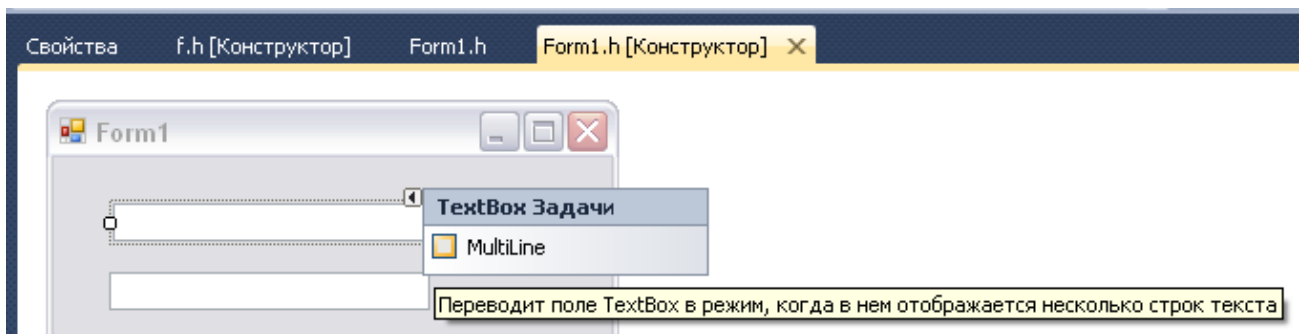
## Лабораторна робота №3

### Налаштування інтерфейсу користувача

#### Теоретичні відомості

Компонент **Label** призначений лише для відображення текстової інформації (редагування тексту не можливо). Задати текст, що відображається в полі компоненту, можна як під час розробки форми, так і під час роботи програми, присвоївши значення властивості **Text**.

Компонент **TextBox** призначений для введення даних з клавіатури. Залежно від налаштування компоненту, в полі редагування можна вводити один або декілька (**Multiline = True**) рядків тексту.



Всі дані ведені у текстове поле сприймаються програмою, як набір символів. Для перетворення символів до дійсного типу використовують методи **ToDouble** або **ToInt32** (цілочисловий тип).

*Методи компоненту TextBox*

<b>AppendText ()</b>	Додає текст до поточного тексту у вікні компоненту
<b>Clear ()</b>	Очищення поля
<b>Focus ()</b>	Встановлення фокуса

#### Завдання для самостійного виконання

**Завдання 1.** Розробіть додаток використовуючи компоненти введення та виведення даних (Label, TextBox). Врахувати перевірку чи заповнено всі необхідні поля.





### Методичні вказівки:

**If (textBox1->Text=="") textBox1->Focus();** //перевірка умови - якщо поле TextBox не заповнено то курсор встановлюється у поле TextBox

**button1 ->Enabled = false;** // кнопка не активна

**button1 ->Visible = true** // кнопка видима

**УВАГА!** Активація видимості та активності кнопки реалізується у події **TextChanged** компоненту **textBox**

**Приклад.** Активації кнопки при введенні даних у текстове поле




```
private: System::Void textBox1_TextChanged(...)
{
    if (textBox1->Text->Length != 0 )
        button1->Enabled = true;
    else
        button1->Enabled =false;
}
```

1. Написати програму визначення вартості придбаного товару. У поле форми ввести кількість одиниць товару, вартість товару - константа. (Кнопка “обчислити” активна лише за умови якщо введено всі необхідні дані.)

2. Обчислити периметр трикутника. (Кнопка “обчислити” видима лише за умови якщо введено всі необхідні дані.)

3. Написати програму переведення метрів у сантиметри. (Якщо одне з полів не заповнено, то виводиться повідомлення про помилку і курсор встановлюється у порожнє поле. )

4. Написати програму знаходження значення виразу  $p = y * x$ . (Якщо одне з полів не заповнено, то виводиться повідомлення про помилку і курсор встановлюється у порожнє поле. )


 Методичні вказівки до задачі 3:

```
{
    int m, s;
    if (textBox1->Text == "") textBox1->Focus();
    else
    {
        m=Convert.ToInt32(textBox1->Text);
        s=100*m;
        label1->Text=Convert.ToString(s);
    }
}
```

### Завдання 2. (Оператор циклу)

1. Написати програму, яка підраховує суму всіх непарних чисел від А до В.
2. Написати програму, яка обчислює суму перших N елементів ряду  $1+1/2+1/3+....$  (кількість вводиться під час роботи програми).

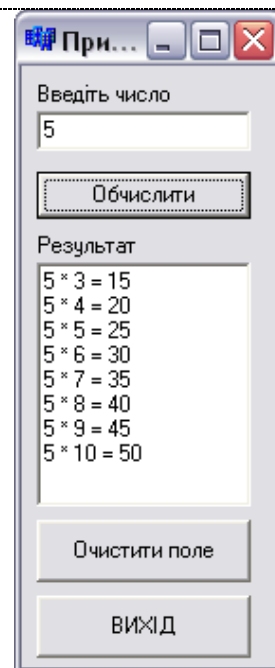
### Завдання 3. Виведіть таблицю множення на довільне число.

 Методичні вказівки:

```
{
    int r;
    int x = (Convert.ToInt32(textBox1->Text));
    for(int i=1;i<=10;i++)
    {
        r=i*x;
        textBox2->AppendText(x+"*"+i+"="+r+"\\r\\n");
    }
}
```

Очистка поля **TextBox** здійснюється методом **Clear()**  
**textBox2->Clear();**

**"\\r\\n"** – перейти на новий рядок та повернути коретку



1. Написати програму, яка виводить значення кубів чисел, що належать проміжку [1, 9] з кроком 2. Оформити виведення у вигляді:  $1^3=1;3^3=9;5^3=125$  і т.д.
2. Написати програму, яка виводить таблицю значень функції  $y = 3x+2$  в діапазоні (-2;2) з кроком 0,5.
3. Написати програму, яка виводить на екран таблицю вартості яблук в діапазоні від 100 г до 1 кг. з кроком 100г.(вартість вводиться під час роботи програми).

## Лабораторна робота №4

### Робота із змінними числових та рядкових типів.

#### Теоретичні відомості

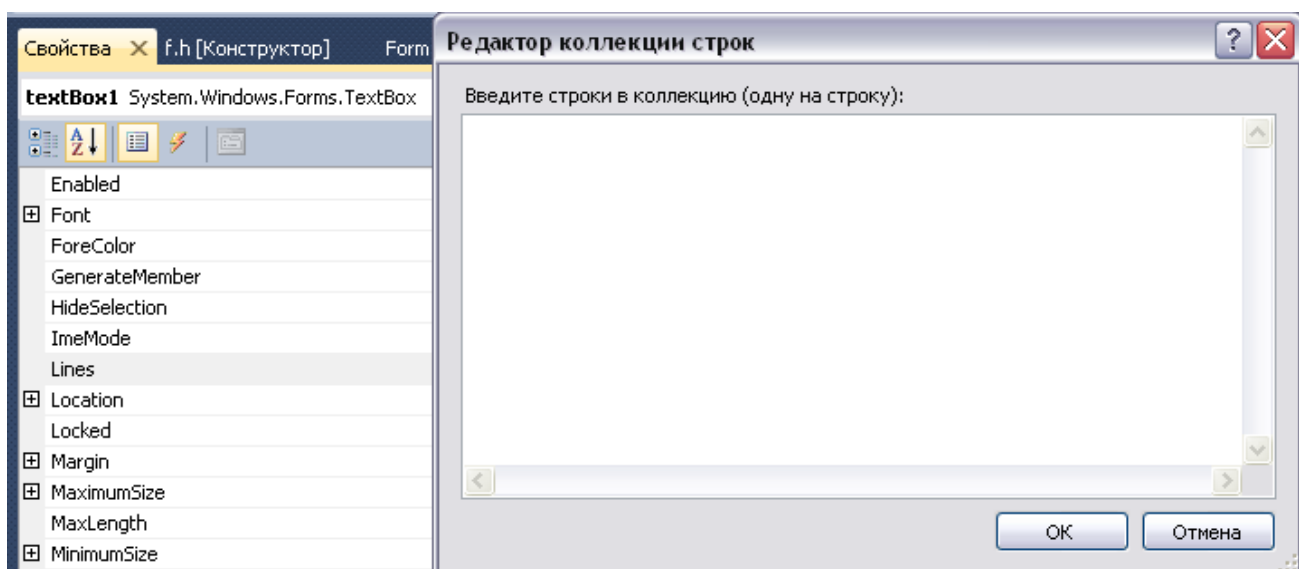
Компонент **TextBox** призначений для введення даних з клавіатури. Залежно від налаштування компоненту, в полі редагування можна вводити один або декілька (**Multiline = True**) рядків тексту.

Всі дані ведені у текстове поле сприймаються програмою, як набір символів. Для перетворення символів до дійсного типу використовують методи **toDouble** або **toInt32** (цілочисловий тип).

**Convert::toDouble** – перетворення змінної до дійсного типу

Властивості компоненту <b>TextBox</b>	
<b>Multiline</b>	Вирішує ( <b>True</b> ) або забороняє ( <b>False</b> ) введення декількох рядків тексту
<b>Lines</b>	Масив рядків, елементи якого містять текст, що знаходиться в полі редагування, якщо компонент знаходиться в режимі <b>MultiLine</b> . Доступ до рядка здійснюється по номеру. Рядки нумеруються з нуля


Lines Массив String[]



## Завдання для самостійного виконання

### Завдання 1.

1. Підрахувати кількість від'ємних чисел із послідовності введених з клавіатури. (Для введення використати компонент **TextBox**.)

 Методичні вказівки:

```
int x,k=0;
for(int i=0;textBox2->Lines[i]!=""; i++)
{
    x=Convert::ToInt32(textBox2->Lines[i]);
    if(x<0)k++;
}
MessageBox::Show("кількість елементів<0 =" +k);
```

2. Підрахувати суму парних чисел із послідовності введених з клавіатури, поки не введено нуль. (Для введення використати компонент **TextBox**.)

```
int x,s=0;
for(int i=0;textBox2->Lines[i]!="";i++)
{
    x=Convert::ToInt32(textBox2->Lines[i]);
    if(x==0)break;
    if(x%2==0) s+=x;
}
MessageBox::Show("сума парних чисел =" +s);
```


3. Написати програму для знаходження суми всіх цілих додатних парних чисел введених з клавіатури.

4. Написати програму для знаходження кількість додатних парних чисел із послідовності введених з клавіатури.

5. Написати програму, яка обчислює добуток всіх парних чисел із послідовності введених з клавіатури.

6. Написати програму, яка визначає максимальне число у введений з клавіатури послідовності цілих додатних чисел. Для завершення вводу послідовності ввести нуль.

## Завдання 2 (Обробка рядків)

 Методичні вказівки:

**Приклад:** Замінити кожен 2 символ рядка на «,» підрахуйте кількість замін

```
String^ s=textBox1->Text;
String^s1=s1+=s[0];
int k=0;
for(int i=1;i<s->Length;i++)
{
    if(i%2!=0) s1+=",";
    else s1+=s[i];
    if (s1[i]=='') k++;
}
label1->Text=s1;
MessageBox::Show("кількість замін = "+k);
```

**Приклад:** Замінити всі літери «a» на прописні «A»

```
String^ s=textBox1->Text;
String^s1=" ";
for(int i=0; i<s->Length; i++)
if(s[i]=='a') s1+="A";
else s1+=s[i];
label1->Text=s1;
```

\*\*\*\*\*

1. Написати програму, яка підраховує кількість ком у рядку.
2. Написати програму, яка замінює всі літера „a” на „1”.
3. Написати програму, яка підраховує кількість слів у рядку.
4. Написати програму, яка визначає, скільки раз у рядку зустрічається літера введена з клавіатури.
5. Написати програму, яка замінює кожен другий символ рядку на „1”.

## Завдання 3. Обробка масивів

В динамічній пам'яті створено об'єкт типу **Random**, який володіє функціями генерації випадкових чисел.

**NextDouble()** – генерує випадкові числа в діапазоні 0.0 – 1.0

**a[i][j]=100.0\*r->NextDouble()** - генерує випадкові дійсні числа в діапазоні 0-100

 Методичні вказівки:

**Приклад:** Вивести у текстовому полі двовимірний масив.

```
//textBox1->Multiline=true
int i,j, a[5][5];
Random^r=gcnew Random;
for(i=0;i<5;i++)
{
    for(j=0;j<5;j++)
    {
        a[i][j]=r->Next(10);
        textBox1->AppendText(a[i][j]+" ");
    }
    textBox1->AppendText("\r\n");
}
```

**Приклад:** Підрахувати кількість нульових елементів двовимірного масиву та замінити всі від’ємні елементи нулями. Введення даних з клавіатури реалізується у компонент **dataGridView1**. Результат заміни у компонент **dataGridView2**.

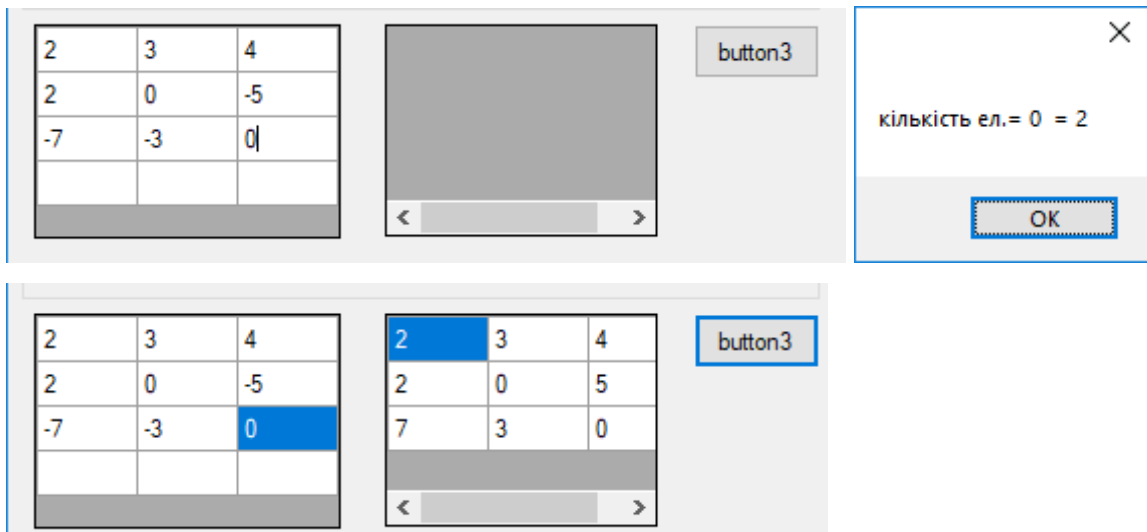
Після розташування на формі компонентів необхідно встановити для них такі властивості:

dataGridView1	AllowUserToAddRows	true
dataGridView1	Columns /Width	Column1/50 Column2/50 Column3/50 Column4/50 Column5/50
dataGridView1	ColomnHeadersVisible	false
dataGridView1	RowHeadersVisible	false
dataGridView2	AllowUserToAddRows	false
dataGridView2	AllowUserToDeleteRows	false
dataGridView2	ReadOnly	true
dataGridView2	Columns /Width	Column1/50 Column2/50 Column3/50 Column4/50 Column5/50
dataGridView2	ColomnHeadersVisible	false
dataGridView2	RowHeadersVisible	false

```

int i,j, a[3][3], k=0;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
    a[i][j]= Convert::ToInt32(dataGridView1->Rows[i]->Cells[j]->Value);
    if(a[i][j]==0)k++;
}
MessageBox::Show("кількість елементів= 0 = "+k);
// заміна та виведення елементів масиву
for(i=0;i<3;i++)
{
    dataGridView2->Rows->Add();
    for(j=0;j<3;j++)
    {
        if (a[i][j]<0)a[i][j]=0;
        dataGridView2->Rows[i]->Cells[j]->Value =Convert::ToString(a[i][j]);
    }
}

```



\*\*\*\*\*

1. Написати програму, яка замінює всі парні елементи масиву на „0”.
2. Написати програму, яка визначає, скільки раз у масиві зустрічається введене з клавіатури число.
3. Написати програму, яка визначає суму елементів, які мають непарні порядкові номери.
4. Написати програму, яка визначає кількість учнів у класі чий зріст вище середнього.
5. Написати програму, яка визначає мінімальний елемент масиву та його порядковий номер.

## Лабораторна робота №5

### Розробка додатків з використанням елементів управління.



#### Теоретичні відомості

Радіокнопки **RadioButton** утворюють групи взаємозв'язаних індикаторів, з яких звичайно може бути вибраний тільки один. Вони використовуються для вибору користувачем однієї з декількох взаємовиключних альтернатив

Окрема радіокнопка **RadioButton** особливого сенсу не має. А радіокнопки мають сенс, коли вони взаємодіють один з одним в групі.

Компонент **GroupBox** (вкладка «*Контейнери*» панелі елементів) – є контейнером, об'єднуючим групу зв'язаних елементів управління, таких, як радіокнопки **RadioButton** або індикатори **CheckBox** і т.д.

Індикатори з прапорцем **CheckBox** використовуються у додатках для того, щоб вмикати та вимикати опції, або для індикації стану. При кожному клацанні користувача на індикаторі його стан змінюється, проходячи в загальному випадку послідовно через три значення: виділення (*поява чорної галочки*), проміжне (*сіре вікно індикатора і сіра галочка*) і не виділене (*порожнє вікно індикатора*). Цим трьом станам відповідають три значення властивості компоненту.

#### Основні властивості компоненту **CheckBox**

**Checked** – властивість, яка визначає чи включено прапорець.

**CheckState** – встановлює 3 вида станів компоненту:

- **Checked** - Прапорець увімкнено;
- **Unchecked** - Прапорець вимкнено
- **Indeterminate** - Невизначений стан (сірий колір прапорця).

**CheckAlign** – властивість, що дозволяє відкривати розгортаючийся список, для вибору розташування прапорця у полі компоненту (9 позицій).

**AutoCheck** - властивість визначає, чи повинно автоматично змінюватися стан прапорця в результаті клацання на його зображенні. За умовчанням значення рівне True.



Стан прапорця змінюється в результаті клацання на його зображенні (якщо значення властивості **AutoCheck = True**). При цьому виникає подія **CheckedChanged**, а потім подія **Click**. Якщо значення властивості **AutoCheck = False**, то в результаті клацання на прапорці виникає подія **Click**, а потім, якщо процедура обробки цієї події змінить стан кнопки, виникає подія **CheckedChanged**.

## Завдання для самостійного виконання

**Завдання.** Розробіть додаток використовуючи елементи управління

1. Розробити програму обчислення вартості розмови по мобільному телефону використовуючи компонент **RadioButton**, за допомогою якого визначається вартість розмови (в середині мережі, інший оператор).

2. Написати програму обчислення вартості вікна за введеною висотою та шириною. Програма повинна містити вибір опції (однокамерний чи двокамерний пакет) та додаткова можливість замовити підвіконня (**CheckBox** і **RadioButton**).



### Методичні вказівки

*// Використання компоненту **radioButton**. Подія **CheckedChanged***

```
private: System::Void radioButton1_CheckedChanged(...)
{
    int a,b,s;
    a=Convert::ToInt32(textBox1->Text);
    b=Convert::ToInt32(textBox2->Text);
    if(radioButton1->Checked)
    {
        s=a+b;
        label1->Text=Convert::ToString(s);
    }
}
```

*// Використання компоненту **checkBox**. Подія **CheckedChanged***

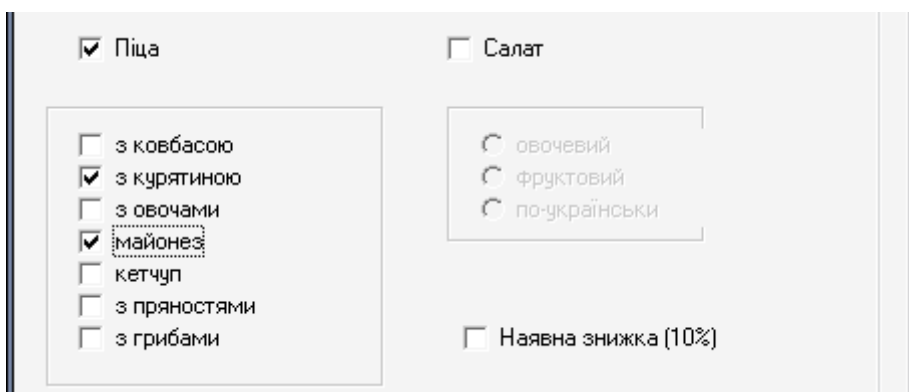
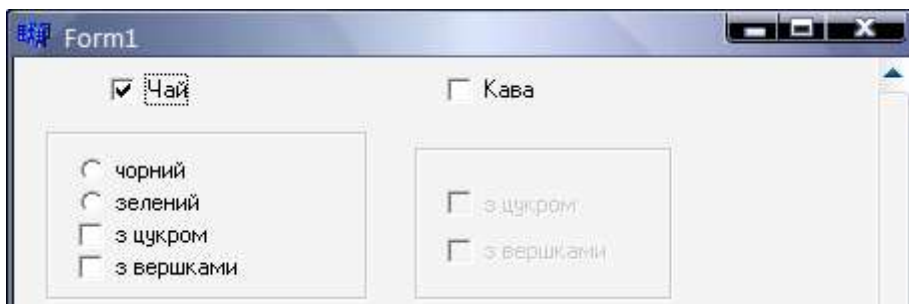
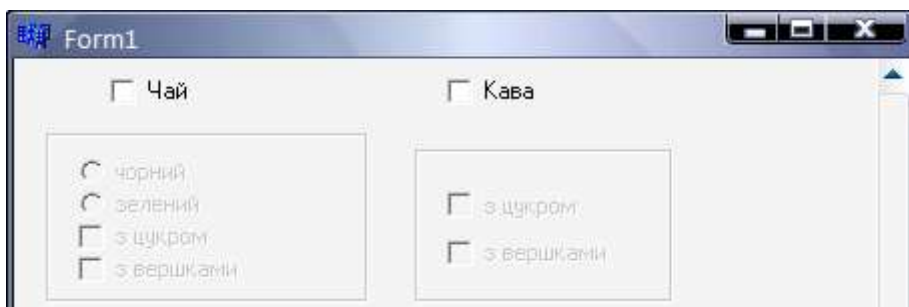
```
private: System::Void checkBox1_CheckedChanged(    )
{
    if (checkBox1->Checked)
        label2->Text="прапорець встановлено";
    else
        label2->Text="прапорець знято";
}
```

3. Розробити програму обчислення вартості замовлення в кафе використовуючи компонент **CheckBox** і **RadioButton**.



- Одна зі страв повинна бути складною (наповнення якої обирає клієнт), компоненти якої неактивні (або невидимі) поки не вибрано назву страви.
- Врахувати можливість відмови від замовлення на певну страву. Вартість замовлення автоматично змінюється при виборі страви, або при відмовленні від неї.

- Враховувати можливість знижки.
- Додати зображення страв, які з'являються при виборі страви.



## Лабораторна робота №6

### Створення головного та контекстному меню додатків



**Завдання 1.** Завантаження зображення в **PictureBox** за допомогою **ComboBox**.

При виборі одного із слів в списку повинна з'являтися картинка, а в **"label"** її назва. Для того що б занести в **"ComboBox"** деякий список, потрібно знайти в панелі властивостей - властивість **"Items"** і написати через **"enter"** список назв зображень



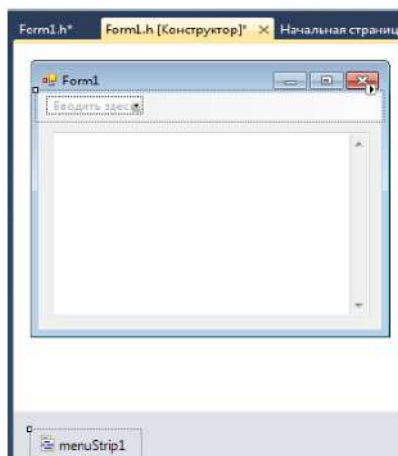
```
#pragma endregion
```

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    this->Text = "Фотогалерея";
    label1->Text = "";
    comboBox1->Text = "Список";
}
```

```
private: System::Void comboBox1_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)
{
    switch (comboBox1->SelectedIndex)
    {
        case 0: pictureBox1->Image=Image::FromFile("d:\\7.0.png");
        label1->Text = "Літо"; break;
        case 1: pictureBox1->Image=Image::FromFile("d:\\7.1.png");
        label1->Text = "Сонце"; break;
        case 2: pictureBox1->Image=Image::FromFile("d:\\7.2.png");
        label1->Text = "Море"; break;
        case 3: pictureBox1->Image=Image::FromFile("d:\\7.3.png");
        label1->Text = "Пляж"; break;
    }
}
```

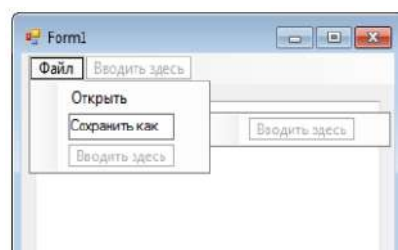
## Завдання 2. Елемент MenuStrip і властивість Anchor в MVS C++

Розглянемо досить важливий і поширений елемент "MenuStrip". "MenuStrip" - це, по суті, випадне меню, в якому є певні пункти. Так само буде розглянута властивість "Anchor" - це властивість, при якій визначається до яких сторін форми буде прив'язаний елемент, якщо до правої, то при збільшенні або зменшенні розміру форми - права сторона "textBox", так само буде збільшуватися або зменшуватися. Якщо поставлена кнопка "button", то вона переміщатиметься за стороною форми, до якої прив'язана. Перенесіть на форму елемент "MenuStrip" і елемент "textBox". У елементу "textBox" застосуйте властивості "Multiline" і "ScrollBars" -> "Vertical"

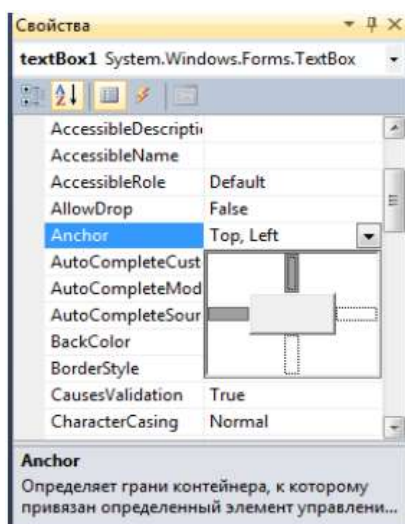


Далі натисніть на текстове поле "Вводите здесь" і напишіть - "Файл":

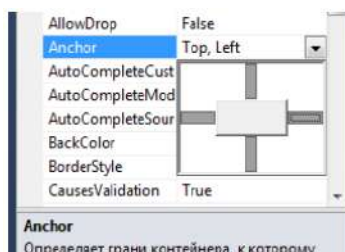
Після цього в нижньому, новому текстовому полі, що з'явилося, напишіть "Відкрити", після чого з'явиться ще одне, в якому напишіть "Зберегти як":



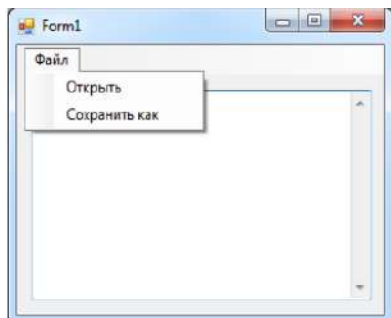
Тепер займемося властивістю "Anchor" - для цього натисніть на "textBox". Після чого на панелі властивостей, розташованій зліва, виберіть властивість "Anchor":



Компонент "textBox" прив'язаний до лівої і верхньої частини форми. Тепер виділіть так само праву і нижню частину, після чого розмір елементу "textBox" змінюватиметься пропорційно розміру форми:



Запустіть програму і перевірте змінені властивості:



**Завдання для самостійного виконання.** Додайте контекстне меню текстового поля.

### Завдання 3. Відкриття і запис текстового файлу в MVS C++

Створимо простий текстового редактора.

У даній програмі ми відкриватимемо вже створений текстовий файл і редагуватимете його або ж самі писати текст і зберігати його, як новий текстовий

файл в потрібну вам папку. Ще одна явна ознака текстового редактора - це якщо ви щось написали і натискаєте хрестик щоб вийти - при цьому програма питає: "Зберегти зміни". Для створення текстового редактора необхідні наступні елементи: "MenuStrip", "textBox", "openFileDialog", "saveFileDialog".



Перетягніть всі ці елементи на форму, назвіть заголовок "MenuStrip" "Файл" створіть в ньому три пункти: "Відкрити", "Зберегти як", "Вихід", прив'яжіть "textBox" до всіх сторін форми, встановіть властивість "Multiline" і "ScrollBars->Vertical".

У кодї програми будуть створені "MyReader" і "MyWriter", за допомогою яких програма читатиме і записуватиме текст у файл. Крім того в кодї створюється кодування, завдяки якому програма розумітиме російський текст. У форми потрібно викликати подію "FormClosing".

**Завдання для самостійного виконання.** Проаналізуйте запропонований код та реалізуйте текстовий редактор з командами відкриття та збереження текстового файлу.

```
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    this->Text = "Текстовий редактор";
    openFileDialog1->FileName = "D:\\ВУЗ\\Text2.txt";
    openFileDialog1->Filter = "Текстові файли (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1->Filter = "Текстові файли (*.txt)|*.txt|All files (*.*)|*.*";
}

private: System::Void открытьToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)
{
    openFileDialog1->ShowDialog();
    if (openFileDialog1->FileName == nullptr) return;
    try
    {
        auto MyReader = gcnew IO::StreamReader(openFileDialog1->FileName, System::Text::Encoding::GetEncoding(1251));
        textBox1->Text= MyReader->ReadToEnd();
        MyReader->Close();
    }
    catch (IO::FileNotFoundException^ Ситуация)
    {
        MessageBox::Show(Ситуация->Message + "\nФайл не найден", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
    }
    catch (Exception^ Ситуация)
    {
        MessageBox::Show(Ситуация->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
    }
}

private: System::Void сохранитьКакToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)
{
    saveFileDialog1->FileName = openFileDialog1->FileName;
    if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) Save();
}
void Save()
{
    try
    {
        // створення екземпляру StreamWriter для запису у файл:
```

```

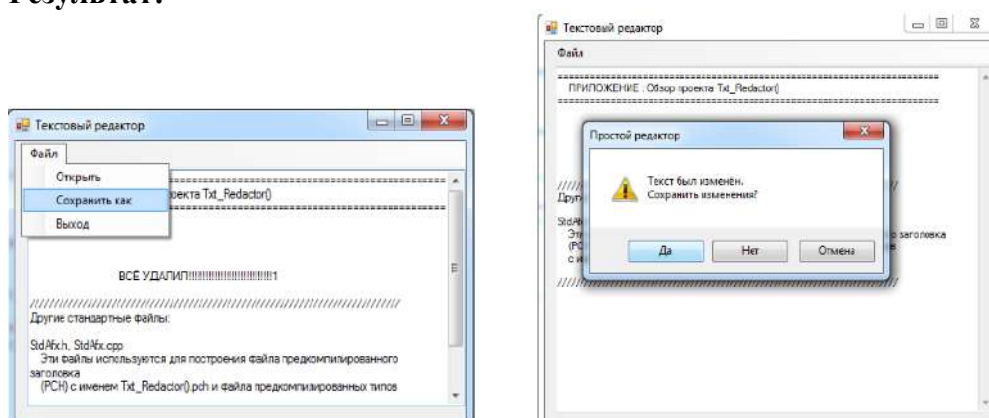
auto MyWriter = gcnew IO::StreamWriter(saveFileDialog1->FileName, false,
System::Text::Encoding::GetEncoding(1251));
MyWriter->Write(textBox1->Text);
MyWriter->Close(); textBox1->Modified = false;
}
catch (Exception^ Ситуация)
{
MessageBox::Show(Ситуация->Message, "Ошибка", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}}

private: System::Void выходToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e)
{
    this->Close();
}

private: System::Void Form1_FormClosing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e)
{
if (textBox1->Modified == false) return;
auto MeBox = MessageBox::Show("Текст було зміне. \n Зберегти?", "Текстовий
редактор", MessageBoxButtons::YesNoCancel, MessageBoxIcon::Exclamation);
if (MeBox == Windows::Forms::DialogResult::No) return;
if (MeBox == Windows::Forms::DialogResult::Cancel) e->Cancel = true;
if (MeBox == Windows::Forms::DialogResult::Yes)
{
if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)
{
Save();
return;
}
else e->Cancel = true;
}}};}

```

## Результат:

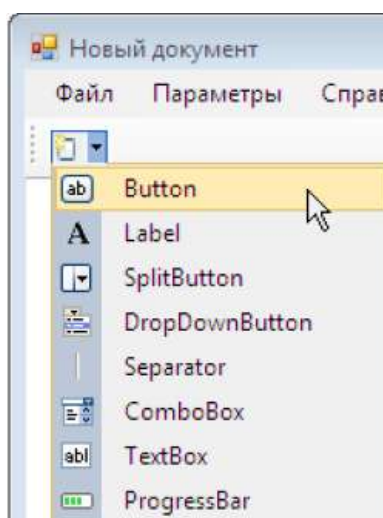


## Лабораторна робота №7

### Компоненти зовнішнього оформлення. Панелі інструментів.

#### Теоретичні відомості

Компонент **ToolStrip** - смуга (панель) інструментів використовується для розміщення в ньому інших компонентів. У панель інструментів можна помістити командну кнопку, поле відображення тексту, поле редагування, список, що розкривається.



#### Властивості компоненту ToolStrip

**Buttons – (Items)** колекція компонентів, що знаходяться у панелі інструментів.

**Dock** – межа батьківського компоненту (форма) до якої прив'язано панель інструментів.

**Visible** – дозволяє приховати або відобразити панель інструментів.

**Enabled** – доступ до компонентів панелі інструментів.

Для того, щоб в панель інструментів додати, наприклад, командну кнопку, треба вибрати компонент **ToolStrip**, клікнути на значку розкриваючогося списку, який відображається в його полі, і вибрати **Button**. Після цього треба виконати налаштування доданої кнопки **toolStripButton** - задати значення властивостей.



## Завдання для самостійного виконання

**Завдання 1.** Модифікуйте програму «Текстовий редактор». Додайте панель інструментів.



### Методичні вказівки

**Приклад** подій, які відповідають за вирівнювання тексту

```
private: System::Void toolStripButton11_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->SelectionAlignment = HorizontalAlignment::Left;
}
private: System::Void toolStripButton12_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->SelectionAlignment = HorizontalAlignment::Center;
}
private: System::Void toolStripButton13_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->SelectionAlignment = HorizontalAlignment::Right;
}
```

**Приклад** подій, які відповідають за відміну останньої дії

```
private: System::Void toolStripButton1_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Undo();
}
private: System::Void toolStripButton2_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Redo();
}
```

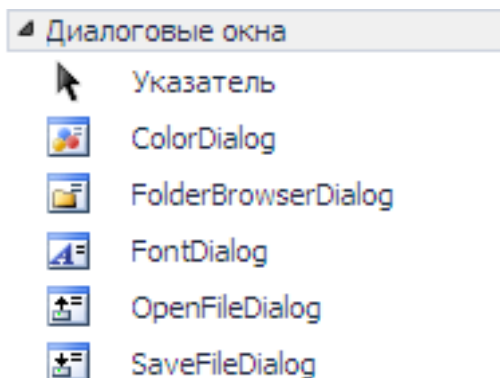
**Приклад** подій, які відповідають за операції (вирізати, копіювати, вставити)

```
private: System::Void вырезатьToolStripButton_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Cut();
}
private: System::Void копироватьToolStripButton_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Copy();
}
private: System::Void вставкаToolStripButton_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Paste();
}
```

## Лабораторна робота №8.

### Розробка діалогових вікон. Системні діалоги

#### Теоретичні відомості



#### Компонент OpenFileDialog

Компонент представляє собою діалогове вікно «Открыть».

При додаванні на форму компонент розташовується у окремій області вікна дизайнера форм. Якщо необхідно відкрити папку, а не файл – використовують клас **FolderBrowserDialog**.

Діалогове вікно для вибору файлу з'являється у режимі виконання додатку при виклику метода **ShowDialog()**. Коли користувач у діалоговому вікні натисне на кнопку «Открыть», метод **ShowDialog()** повертає значення **DialogResult**, яке порівнюється зі значенням такої ж властивості у форми. Для форми значення властивості =OK.

Основні властивості компоненту:

**Multiselect** – дозволяє вибрати групу файлів.

**Filter** – умовна фільтрація файлу.

Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*

**ShowReadOnly** – дає можливість з'являтися індикатору поряд з файлом, якщо він має тип «тільки для читання»

#### Компонент SaveFileDialog.

Компонент для відображення діалогового вікна «**Сохранить**» та збереження файлу. Але компонент дає лише шлях до місця розташування файлу, написання коду, який відповідає за збереження даних відводиться програмісту.

Діалогове вікно з'являється у режимі виконання додатку при виклику метода **ShowDialog()**. Більшість властивостей компоненту аналогічна компоненту **OpenFileDialog**. Для відображення вікна «Сохранить как» необхідно значення властивості **OverwritePromp = true**.

Коли користувач обирає ім'я файлу та натискає кнопку збереження метод **ShowDialog()** заносить у властивість **FileName** компоненту ім'я файлу та шлях до нього. При цьому жодного перезапису файлу не відбувається.

**Компонент ColorDialog** - викликає діалогове вікно вибору кольору. Значення кольору записується у властивість **Color**.

**Компонент FontDialog** – виклик вікна налаштування параметрів шрифту. Обрані параметри присвоюються властивості компоненту **Font**.


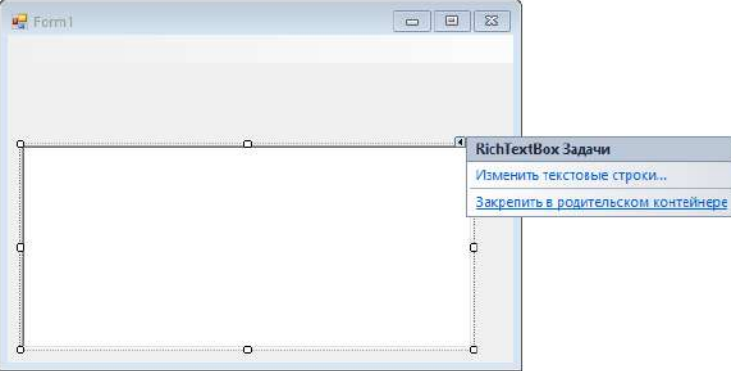


**Компонент PrintDialog** – відкриває діалогове вікно налаштування параметрів друку

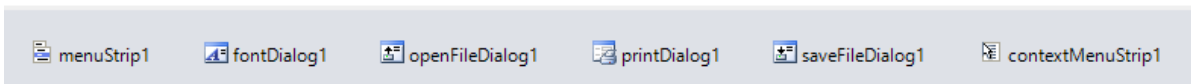
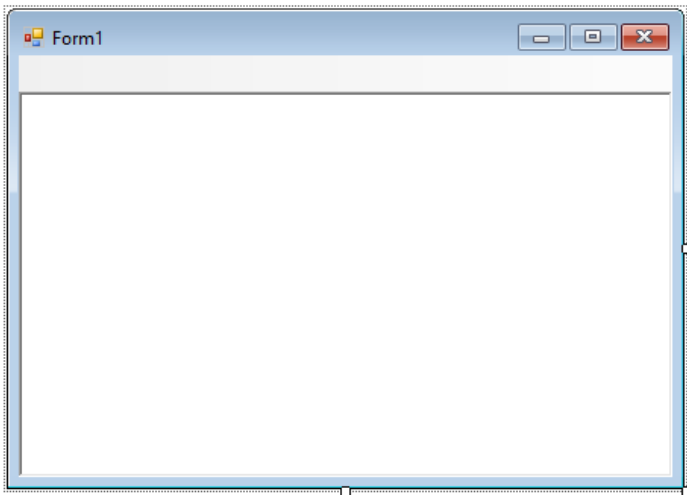
### Завдання для самостійного виконання

**Завдання 1.** Модифікуйте програму «Текстовий редактор». Проаналізуйте попередній програмний код.

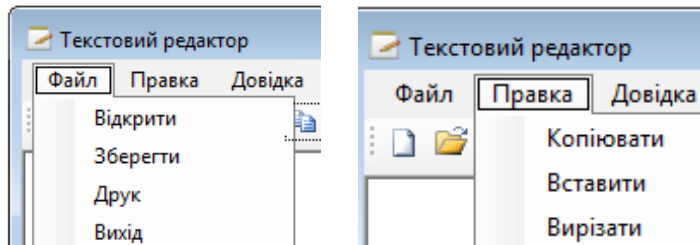
1. Додайте на нову форму компоненти

 <b>OpenFileDialog</b>	openFileDialog не візуальний компонент представляє собою діалогове вікно « <b>Открыть</b> ».
 <b>SaveFileDialog</b>	Компонент для відображення діалогового вікна « <b>Сохранить</b> » та збереження файлу. Але компонент дає лише шлях до місця розташування файлу, написання коду, який відповідає за збереження даних відводиться програмісту.
 <b>FontDialog</b>	Компонент <b>FontDialog</b> – виклик вікна налаштування параметрів шрифту. Обрані параметри присвоюються властивості компоненту <b>Font</b> .
 <b>PrintDialog</b>	Компонент <b>PrintDialog</b> – відкриває діалогове вікно налаштування параметрів друку

 RichTextBox	 <p>Компонент необхідно закріпити у батьківському контейнері</p>
 MenuStrip	Головне меню
 ContextMenuStrip	Контекстне меню (застосувати до компоненту richTextBox)



2. Використовуючи компонент **MenuStrip** створіть головне меню програми.



3. Функція збереження тексту створюється програмістом.

```

void save()
{
    auto read = gcnew IO::StreamWriter(saveFileDialog1->FileName, false,
    System::Text::Encoding::GetEncoding(1251));
    read->Write(richTextBox1->Text);
    read->Close();
    richTextBox1->Modified = false;
}

```

#### 4. ПУНКТ МЕНЮ «Зберегти»

```

saveFileDialog1->FileName = openFileDialog1->FileName;
saveFileDialog1->Filter = "txt files (*.txt)|*.txt|rtf files
(*.rtf)|*.rtf|All files (*.*)|*.*";
if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)
save();

```

#### 5. ПУНКТ МЕНЮ «Відкрити»

```

openFileDialog1->Filter = "txt files (*.txt)|*.txt|rtf files
(*.rtf)|*.rtf|All files (*.*)|*.*";
if( openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK){
System::Text::Encoding^kod=System::Text::Encoding::GetEncoding(1251);
auto read= gcnew IO::StreamReader(openFileDialog1->FileName,kod);
richTextBox1->Text=read->ReadToEnd();}
this->Text+=(openFileDialog1->FileName);

```

#### 6. ПУНКТ МЕНЮ «Друк»

```

printDialog1->ShowDialog();

```

#### 7. Пункти меню «Копіювати», «Вирізати», «Вставити».

```

richTextBox1->Copy();
richTextBox1->Cut();
richTextBox1->Paste();

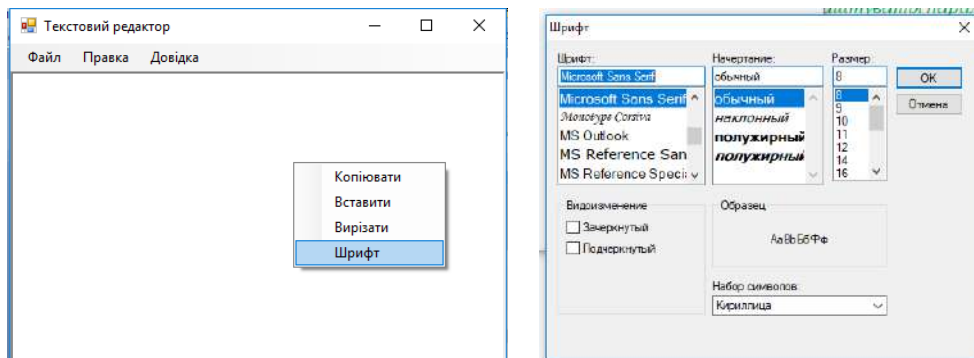
```


8. Додайте пункт меню «Довідка» з інформацією про розробника програмного продукту.

9. Додайте контекстне меню. Пункти меню «Копіювати», «Вирізати», «Вставити», «Шрифт».

//відкриває діалогове вікно налаштування параметрів шрифту

```
this->fontDialog1->ShowDialog();  
richTextBox1->SelectionFont=fontDialog1->Font;
```



10. Додаткове завдання. Додати панель інструментів зі стандартним набором кНОПОК.  ToolStrip



## Лабораторна робота №9

### Особливості роботи з графічними примітивами

#### Теоретичні відомості

Відображення графіки забезпечує компонент PictureBox. Графічна поверхня компоненту PictureBox є об'єктом Graphics, методи якого і забезпечують виведення графіки.

Методи малювання графічних примітивів використовують *олівці* і *пензлі*. Олівець (об'єкт Pen) визначає вид лінії, пензель (об'єкт Brush) - вид заливки області. Побудова графічних примітивів на графічній поверхні (Graphics) виконують відповідні методи.

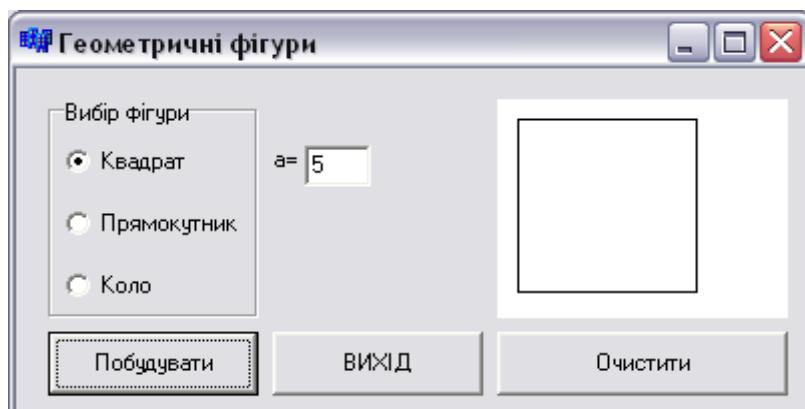
Метод	Дія
DrawLine(Pen, x1, y1, x2, y2), DrawLine(Pen, p1, p2)	Пряма
DrawRectangle(Pen, x, y, w, h)	Прямокутник
FillRectangle(Brush, x, y, w, h)	Зафарбований прямокутник
DrawEllipse(Pen, x, y, w, h)	Еліпс
FillEllipse(Brush, x, y, w, h)	Зафарбований еліпс
DrawPolygon(Pen, P)	Багатокутник
FillPolygon(Brush, P)	Зафарбований багатокутник
DrawString(str, Font, Brush, x, y)	Текст

#### **Приклад.** *Зображення прапора України.*

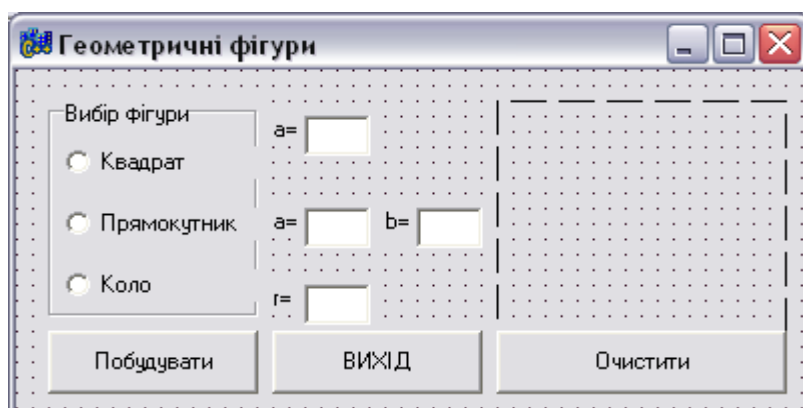
```
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
    int x,y; // лівий верхній кут
    int w,h; // ширина та висота смуги
    x = 10;
    y = 10;
    w = 100;
    h = 30;
    e->Graphics->FillRectangle(System::Drawing::Brushes::Blue,x,y,w,h);
    y += h;
    e->Graphics->FillRectangle(System::Drawing::Brushes::Yellow,x,y,w,h);
    // Підпис. Використовується шрифт, заданий властивістю Font форми.
    e->Graphics->DrawString("УКРАЇНА",this->Font,System::Drawing::Brushes::Black,
    10, 80);
}
```

## Завдання для самостійного виконання

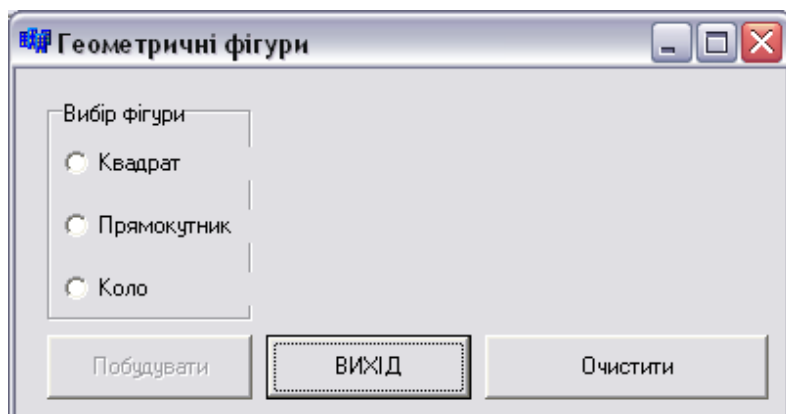
**Завдання 1.** Написати програму для демонстрації побудови геометричних фігур. Фігура виводиться у компоненті PictureBox.



### Методичні вказівки:

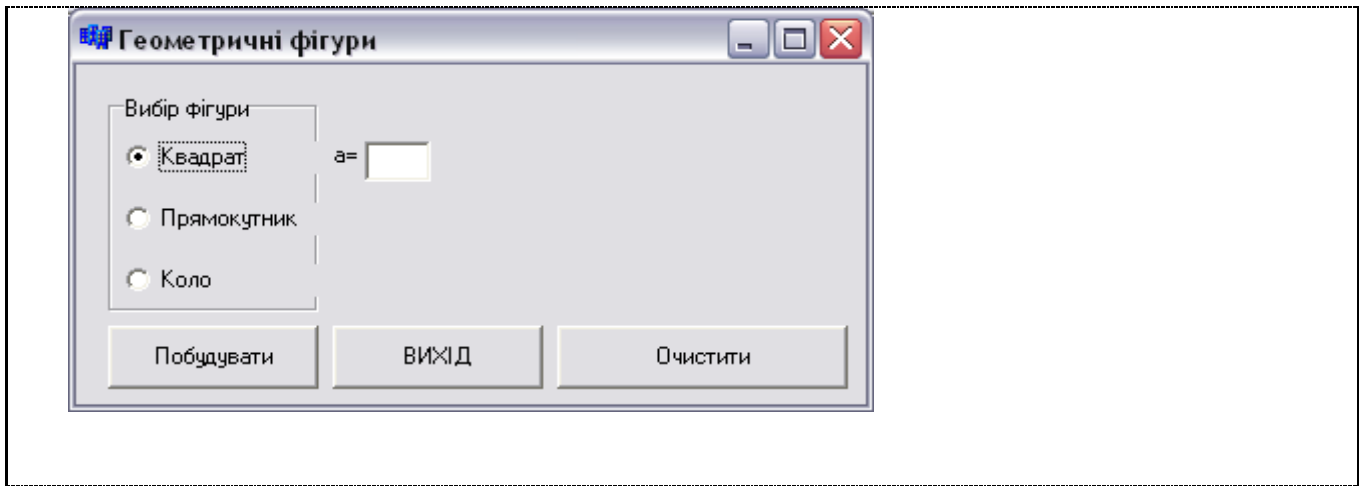


1. При завантаженні кнопка **Побудувати** – неактивна (**Enabled = False**) а поля для введення параметрів – невидимі (**Visible = False**).

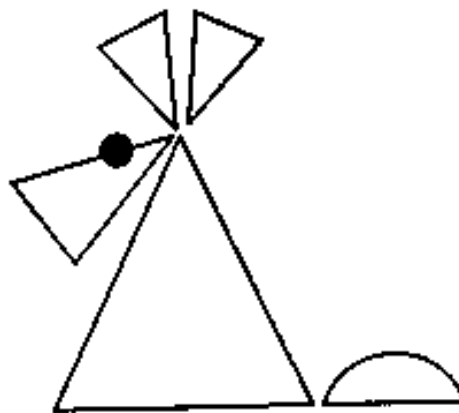
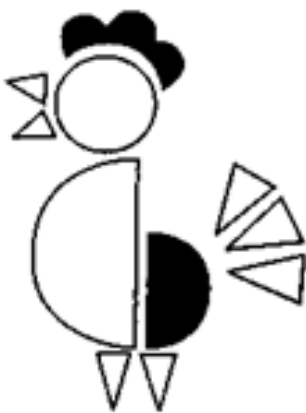


2. Після вибору геометричної фігури стають видимими поля для введення відповідних параметрів.





**Завдання 2.** Побудуйте малюнки використовуючи графічні примітиви.



Методичні вказівки:

Метод DrawPolygon креслить багатокутник (контур).

**DrawPolygon(aPen, p)**

Зафарбований багатокутник малює метод FillPolygon.

**FillPolygon(aBrush, p)**

```
array<Point>^ p;
p = gnew array<Point>(5);
p[0].X = 10; p[0].Y = 30;
p[1].X = 10; p[1].Y = 10;
p[2].X = 30; p[2].Y = 20;
p[3].X = 50; p[3].Y = 10;
p[4].X = 50; p[4].Y = 30;
e->Graphics->FillPolygon(Brushes::Gold, p);
e->Graphics->DrawPolygon(Pens::Black,p);
```

## Лабораторна робота №10

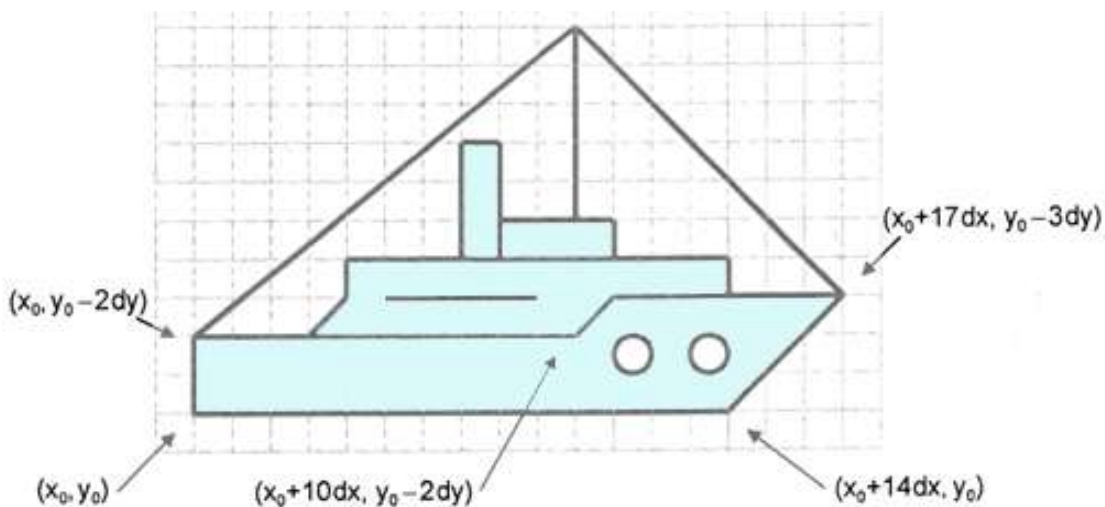
### Метод базової точки

#### Теоретичні відомості

При програмуванні складних зображень, що складаються з безлічі елементів, використовується метод, який називається методом базової точки. Суть цього методу полягає в наступному:

1. Вибирається деяка точка зображення, яка береться за базу.
2. Координати інших точок відлічуються від базової точки.
3. Якщо координати точок зображення відлічувати від базової у відносних одиницях, а не у пікселях, то забезпечується можливість масштабування зображення.

Наприклад зображення кораблика. Базовою - є точка з координатами  $(X_0, Y_0)$ . Координати інших точок відлічуються саме від цієї точки.



$dx, dy$  – масштабування зображення (задається у пікселях.)

*Приклад програми побудови фрагменту кораблика методом базової точки*

```
int X = 70, Y = 100; //базова точка  $X_0, Y_0$ 
int dx=10,dy=10; // крок сітки (масштаб)
array<Point>^ p1;
array<Point>^ p2;
p1 = gcnew array<Point>(7);
p2 = gcnew array<Point>(8);
// координати точок корпусу
p1[0].X = X; p1[0].Y = Y;
```

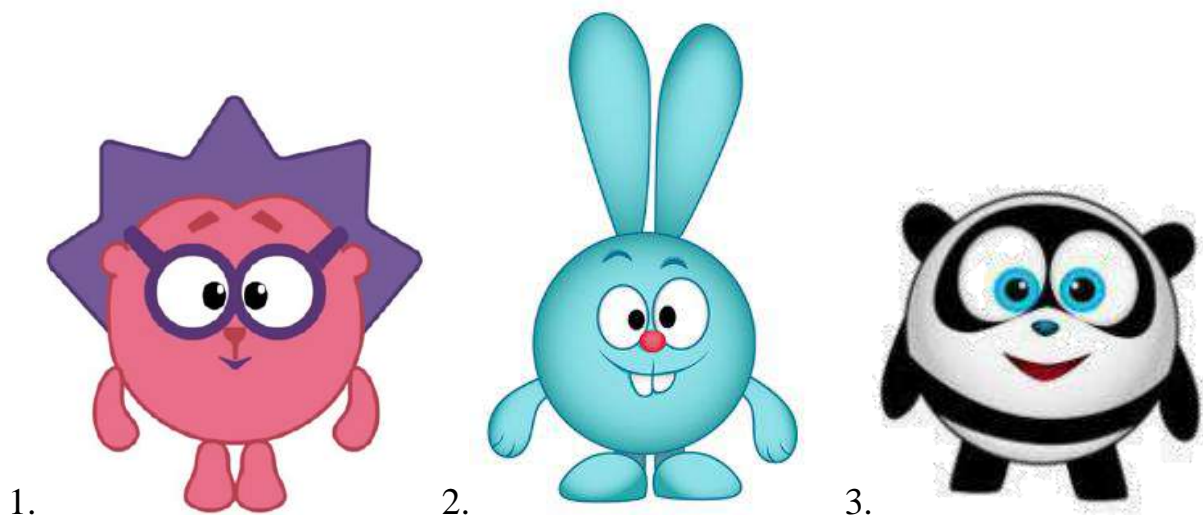
```

p1[1].X = X; p1[1].Y = Y-2*dy;
p1[2].X = X+10*dx; p1[2].Y = Y-2*dy;
p1[3].X = X+11*dx; p1[3].Y = Y-3*dy;
p1[4].X = X+17*dx; p1[4].Y = Y-3*dy;
p1[5].X = X+14*dx; p1[5].Y = Y;
p1[6].X = X; p1[6].Y = Y;
e->Graphics->DrawPolygon(Pens::Black, p1);
// координати точок надбудови
p2[0].X = X+3*dx; p2[0].Y = Y-2*dy;
p2[1].X = X+4*dx; p2[1].Y = Y-3*dy;
p2[2].X = X+4*dx; p2[2].Y = Y-4*dy;
p2[3].X = X+13*dx; p2[3].Y = Y-4*dy;
p2[4].X = X+13*dx; p2[4].Y = Y-3*dy;
p2[5].X = X+11*dx; p2[5].Y = Y-3*dy;
p2[6].X = X+10*dx; p2[6].Y = Y-2*dy;
p2[7].X = X+3*dx; p2[7].Y = Y-2*dy;
e->Graphics->DrawPolygon(Pens::Black, p2);

```

## Завдання для самостійного виконання

**Завдання 1.** Побудуйте малюнок використовуючи олівець програміста та штриховий чи текстурний пензель (за вибором).



### Методичні вказівки:

**Олівець програміста** - це об'єкт типу **Pen**, властивості якого визначають вид лінії, що малюється олівцем. Колір, ширину лінії і стиль олівця, створеного програмістом, можна змінити. Щоб це зробити, треба встановити значення відповідної властивості.

```

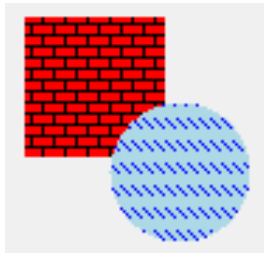
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
System::Drawing::Pen^ aPen; // олівець
// створити червоний «товстий» олівець
aPen = gcnew System::Drawing::Pen(Color::Red,2);
e->Graphics->DrawRectangle(aPen,10,10,100,100);
// створити зелений олівець товщиною 4 пікселя
aPen->Width = 4;
aPen->Color = Color::Green;
....
// стиль лінії - пунктирний
aPen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dash;
....
}

```

**Штриховий пензель (HatchBrush)** зафарбовує область шляхом штрихування.

Область може бути заштрихована горизонтальними, вертикальними або лініями під кутом різного стилю і товщини.

### *Приклад застосування штрихового пензля*



```

// необхідно підключити
using namespace System::Drawing::Drawing2D;
так як HatchBrush - це клас простору імен
System.Drawing.Drawing2D

HatchBrush ^myBrush=gcnew
HatchBrush(HatchStyle::HorizontalBrick, Color::Black,
Color::Red);
e->Graphics->FillRectangle(myBrush, 70, 70, 50, 50);

HatchBrush ^fBrush=gcnew
HatchBrush(HatchStyle::DashedDownwardDiagonal, Color::Blue,Color:
:LightBlue);
e->Graphics->FillEllipse(fBrush, 100, 100, 50, 50);

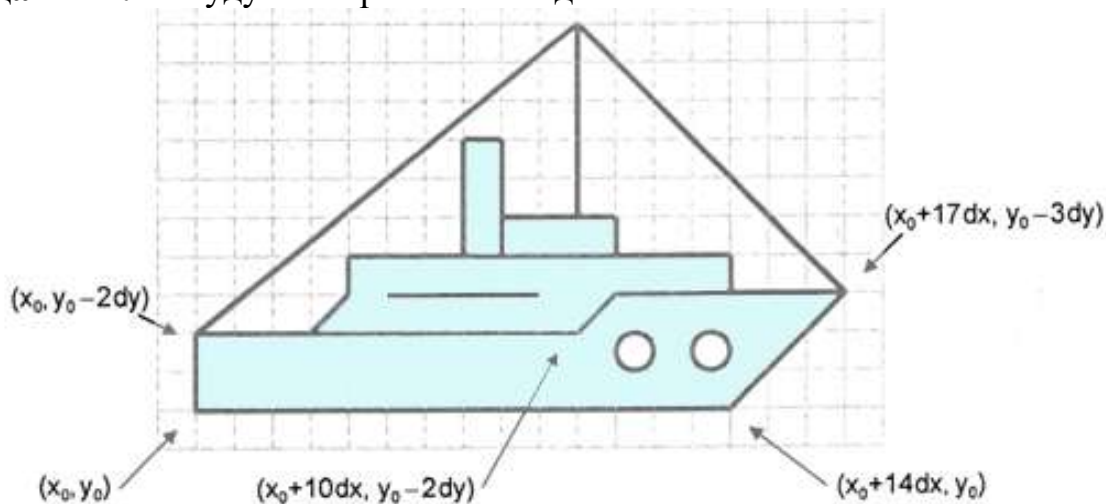
```

**Текстурний пензель (TextureBrush)** є малюнком, який звичайно завантажується під час роботи програми з файлу (bmp, jpg або gif) або з ресурсу.

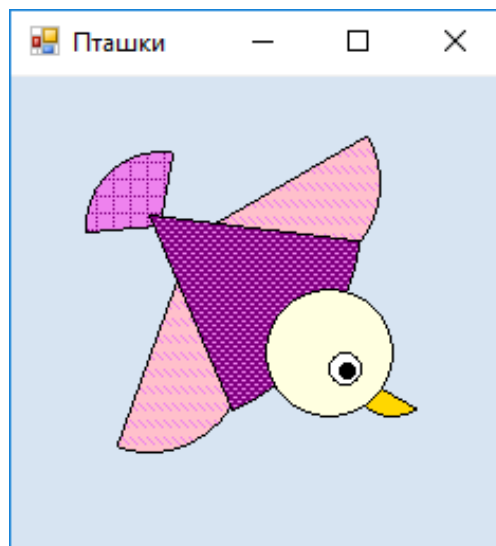
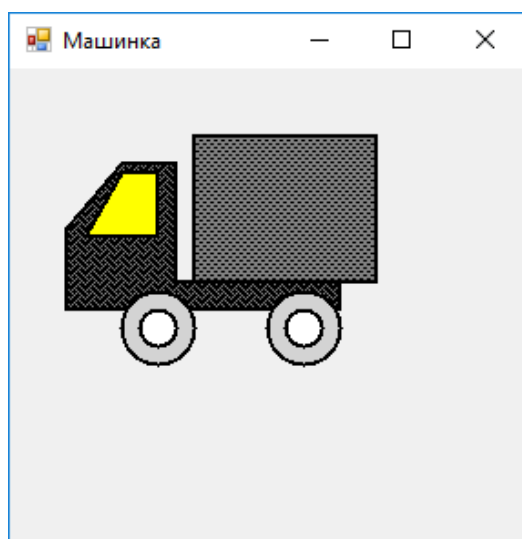
Зафарбовування області виконується шляхом дублювання малюнка у середині області.

```
TextureBrush^ paintBrush=gcnw  
TextureBrush(Image::FromFile("d:p.jpg"));  
e->Graphics->FillRectangle(paintBrush, 150,150,150, 150);
```

**Завдання 2.** Побудуйте пароплав методом базової точки.



**Завдання 3.** Побудуйте зображення методом базової точки (за вибором).



## Лабораторна робота №11

### Створення анімації засобами мови C++

#### Теоретичні відомості

Під анімацією звичайно розуміється рухомий малюнок. Підбором часу між виведенням і видаленням малюнка, а також відстані між старим і новим положенням малюнка (крок переміщення), можна досягти того, що у спостерігача складатиметься враження, що малюнок рівномірно рухається по екрану.

Компонент **Timer**, який використовується для генерації послідовності подій, функція обробки яких забезпечує виведення і видалення малюнка.

Компонент **Timer** генерує подію **Tick**. Період виникнення події **Tick** вимірюється в мілісекундах і визначається значенням властивості **Interval**. Слід звернути увагу на властивість **Enabled**. Воно дає можливість програмі "запустити" або "зупинити" таймер. Якщо значення властивості **Enabled = False**, та подія **Tick** не виникає.

*Приклад програми, яка імітує рух геометричної фігури (червоний квадрат)*

```
namespace anima {
...
using namespace System::Drawing;
int x=50; // оголошення глобальної змінної
...
private: System::Void pictureBox1_Paint (...)
{
e->Graphics->FillRectangle(Brushes::Red,x,70,50,50);
}

private: System::Void timer1_Tick(...)
{
x+=3;
pictureBox1->Refresh();
}
```

#### **Завдання для самостійного виконання**

**Завдання 1.** Створіть анімацію використавши кольорові малюнки створені методом базової точки на попередній лабораторній роботі.

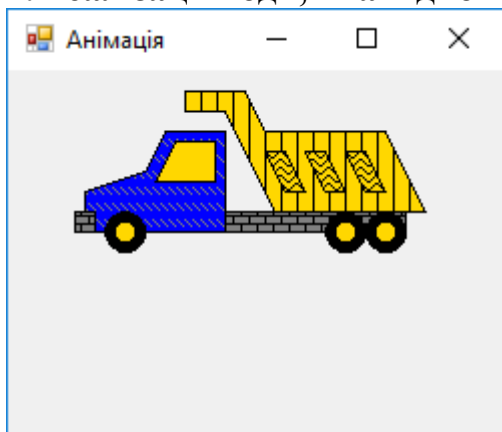
Реалізувати побудову об'єкту у вигляді функції.



Методичні вказівки:

**1. Оголошення глобальної змінної, яка визначає початок руху об'єкту**  
`int x=210;`

**2. Реалізації події, яка відповідає за побудову об'єкту**



```
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
    Pen^ dPen;
    dPen=gcnew Pen(Color::Black,1);

    HatchBrush ^aBrush=gcnew
HatchBrush(HatchStyle::HorizontalBrick,Color::Black,Color::Gray);
    HatchBrush ^bBrush=gcnew
HatchBrush(HatchStyle::DashedDownwardDiagonal,Color::Gray,Color::Blue);
    HatchBrush ^cBrush=gcnew
HatchBrush(HatchStyle::ZigZag,Color::Black,Color::Gold);
    HatchBrush ^dBrush=gcnew
HatchBrush(HatchStyle::Vertical,Color::Black,Color::Gold);

    array<Point>^ kabina;
    kabina = gcnew array<Point>(6);
    kabina[0].X = x+20; kabina[0].Y =80;
    kabina[1].X = x+20; kabina[1].Y =60;
    kabina[2].X = x+50; kabina[2].Y =50;
    kabina[3].X = x+60; kabina[3].Y =30;
    kabina[4].X = x+90; kabina[4].Y =30;
    kabina[5].X = x+90; kabina[5].Y =80;

    array<Point>^ vikno;
    vikno = gcnew array<Point>(4);
    vikno[0].X = x+55; vikno[0].Y =55;
    vikno[1].X = x+65; vikno[1].Y =35;
    vikno[2].X = x+85; vikno[2].Y =35;
    vikno[3].X = x+85; vikno[3].Y =55;

    array<Point>^ rama;
```

```

rama = gcnew array<Point>(4);
rama[0].X = x+90; rama[0].Y =80;
rama[1].X = x+90; rama[1].Y =70;
rama[2].X
rama[3].X = x+180; rama[3].Y =80;

```

```

array<Point>^ kuzov;
kuzov = gcnew array<Point>(8);
kuzov[0].X = x+70; kuzov[0].Y =20;
kuzov[1].X = x+70; kuzov[1].Y =10;
kuzov[2].X = x+100; kuzov[2].Y =10;
kuzov[3].X = x+110; kuzov[3].Y =30;
kuzov[4].X = x+170; kuzov[4].Y =30;
kuzov[5].X = x+190; kuzov[5].Y =70;
kuzov[6].X = x+115; kuzov[6].Y =70;
kuzov[7].X = x+90; kuzov[7].Y =20;

```

```

array<Point>^ bumper;
bumper = gcnew array<Point>(4);
bumper[0].X = x+15; bumper[0].Y =80;
bumper[1].X = x+15; bumper[1].Y =70;
bumper[2].X = x+25; bumper[2].Y =70;
bumper[3].X = x+25; bumper[3].Y =80;

```

```

array<Point>^ kuzov1;
kuzov1 = gcnew array<Point>(4);
kuzov1[0].X = x+120; kuzov1[0].Y =60;
kuzov1[1].X = x+110; kuzov1[1].Y =40;
kuzov1[2].X = x+120; kuzov1[2].Y =40;
kuzov1[3].X = x+130; kuzov1[3].Y =60;

```

```

array<Point>^ kuzov2;
kuzov2 = gcnew array<Point>(4);
    kuzov2[0].X = x+140; kuzov2[0].Y =60;
    kuzov2[1].X = x+130; kuzov2[1].Y =40;
kuzov2[2].X = x+140; kuzov2[2].Y =40;
kuzov2[3].X = x+150; kuzov2[3].Y =60;

```

```

array<Point>^ kuzov3;
kuzov3 = gcnew array<Point>(4);
kuzov3[0].X = x+160; kuzov3[0].Y =60;
kuzov3[1].X = x+150; kuzov3[1].Y =40;
kuzov3[2].X = x+160; kuzov3[2].Y =40;
kuzov3[3].X = x+170; kuzov3[3].Y =60;

```

```

e->Graphics->FillPolygon(bBrush, kabina);
e->Graphics->DrawPolygon(dPen, kabina);
e->Graphics->FillPolygon(Brushes::Gold, vikno);
e->Graphics->DrawPolygon(dPen, vikno);
e->Graphics->FillPolygon(aBrush, rama);
e->Graphics->DrawPolygon(dPen, rama);
e->Graphics->FillPolygon(dBrush, kuzov);

```



```

e->Graphics->DrawPolygon(dPen, kuzov);
e->Graphics->FillPolygon(aBrush, bumper);
e->Graphics->DrawPolygon(dPen, bumper);
e->Graphics->FillEllipse(Brushes::Black, x+30, 70, 20, 20);
e->Graphics->DrawEllipse(dPen, x+30, 70, 20, 20);
e->Graphics->FillEllipse(Brushes::Gold, x+35, 75, 10, 10);
e->Graphics->DrawEllipse(dPen, x+35, 75, 10, 10);
e->Graphics->FillEllipse(Brushes::Black, x+140, 70, 20, 20);
e->Graphics->DrawEllipse(dPen, x+140, 70, 20, 20);
e->Graphics->FillEllipse(Brushes::Gold, x+145, 75, 10, 10);
e->Graphics->DrawEllipse(dPen, x+145, 75, 10, 10);
e->Graphics->FillEllipse(Brushes::Black, x+160, 70, 20, 20);
e->Graphics->DrawEllipse(dPen, x+160, 70, 20, 20);
e->Graphics->FillEllipse(Brushes::Gold, x+165, 75, 10, 10);
e->Graphics->DrawEllipse(dPen, x+165, 75, 10, 10);
e->Graphics->FillPolygon(cBrush, kuzov1);
e->Graphics->DrawPolygon(dPen, kuzov1);
e->Graphics->FillPolygon(cBrush, kuzov2);
e->Graphics->DrawPolygon(dPen, kuzov2);
e->Graphics->FillPolygon(cBrush, kuzov3);
e->Graphics->DrawPolygon(dPen, kuzov3);
}

```

### 3. Організація компоненту для створення ефекту анімації

```

private: System::Void timer1_Tick(System::Object^ sender,
System::EventArgs^ e)
{
x-=3;
if(x== -210) x=210;
pictureBox1->Refresh();
}

```

# ПРАКТИЧНІ РОБОТИ

---

## Практична робота №1 Організація проекту. Відлагодження додатку

---

### Теоретичні відомості

Успішне завершення процесу побудови не означає, що в програмі немає помилок. Переконалися, що програма працює правильно, можна тільки в процесі перевірки її працездатності - тобто *тестуванням*.

Якщо програма працює коректно тільки на деякому обмеженому наборі початкових даних, це свідчить про те, що в програмі є алгоритмічні помилки. Процес пошуку і усунення помилок називається *відлагодженням*.

Помилки, які можуть бути в програмі, прийнято ділити на три групи: синтаксичні; помилки часу виконання; алгоритмічні.

У програмі під час її роботи можуть виникати виключення (помилки), зокрема внаслідок дій користувача. *Наприклад, може ввести невірні дані або, що буває досить часто, видалити потрібний програмі файл.*

Порушення в роботі програми називається *виключенням*. Стандартну обробку виключень, яка в загальному випадку полягає у відображенні повідомлення про помилку, бере на себе код, що автоматично додається у виконувану програму. Разом з тим програміст може (і повинен) забезпечити явну обробку виключень. Для цього в текст програми необхідно помістити інструкції, що забезпечують обробку можливих виключень.

Інструкція обробки виключення в загальному вигляді виглядає так:

**try**

{ */\*фрагмент програми у якій може виникнути непередбачувана ситуація\*/* }

**[catch** (оголошення виключення)

{ */\*оператори для обробки виключення, яке виникло в try-блоці\*/* }]

**throw** вираз

Де:

**try** — ключове слово, що показує, що далі слідує інструкції, при виконанні яких можливо виникнення виключень, і що обробку цих виключень бере на себе програма;

**catch** — ключове слово, що позначає початок секції обробки виключення вказаного типу. Інструкції цієї секції будуть виконані при виникненні виключення вказаного типу.

**throw** (кинути) - ключове слово, що "створює" виключення.

#### Виключення і причини, які можуть привести до їх виникнення.

Тип виключення	Виключення, причина
System::FormatException	Помилка формату. Виникає при виконанні перетворення, якщо перетворювана величина не може бути приведена до необхідного вигляду. Найчастіше виникає при спробі перетворити рядок символів в число, якщо рядок містить невірні символи. <i>Наприклад, при перетворенні рядка в дробове значення, якщо як десятковий роздільник замість коми поставлена крапка</i>
System::IndexOutOfRangeException	Вихід значення індексу за межу області допустимого значення. Виникає при зверненні до неіснуючого елемента масиву
System::IO::FileNotFoundException	Немає файлу. Причина - відсутність необхідного файлу у вказаному каталозі
System::IO::DirectoryNotFoundException	Немає каталога. Причина - відсутність необхідного каталогу

#### ПРИКЛАД:

```
try
{
    a=Convert::ToInt32(textBox1->Text);
    b=Convert::ToInt32(textBox2->Text);
    c=a/b;
    label2->Text=Convert::ToString(c);
}
catch ( System::DivideByZeroException^ e)
{
    MessageBox::Show("Помилка:\n" + e->Message, "Calc", MessageBoxButtons::OK,
    MessageBoxIcon::Error);
}
catch (...)
{
    MessageBox::Show("Помилка");
}
```

## Завдання для самостійного виконання

---

### Завдання 1.

1. Визначити за двома кутами, чи є трикутник прямокутним.
2. Задані координати точки  $x$  та  $y$ . Визначте її розташування в декартовій прямокутній системі координат.
3. Дано натуральне число  $n$ . Знайти першу цифру числа  $n$ .
4. Дано натуральне число  $n$ . Чому дорівнює сума його цифр?
5. Підрахуйте кількість різних літер в слові.
6. Знайдіть у тексті, введеному у поле речення та сформууйте з них окремі рядки.
7. Замініть всі пробіли в тексті на символ «\*» та підрахуйте їх кількість.
8. Додати в рядок пробіли після пунктуаційних знаків, якщо їх там немає.
9. Визначте довжину найкоротшого в рядку слова.
10. Підрахувати кількість чисел у введеному з клавіатури тексті.

### Завдання 2. Напишіть програму міні-калькулятор на 4 арифметичні дії.

Врахуйте перевірку на коректність введення даних (*Обмеження на введення символів та другої коми для дійсних чисел*).

*Рекомендації до виконання // обмеження на введення символів*

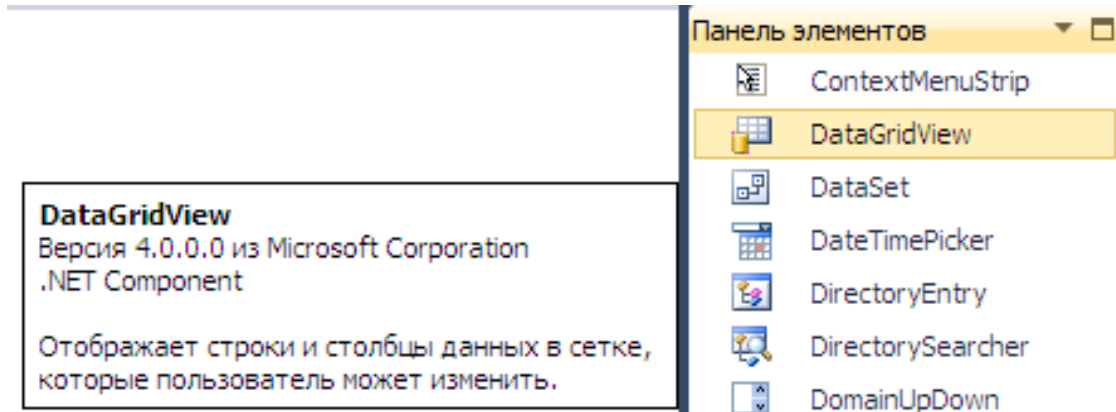
```
try
{
int s = Convert.ToInt32(textBox1->Text);
}
catch (System.FormatException^e)
{
MessageBox.Show("Ви ввели символ! Введіть цифру");
}
textBox1->Clear();
```

## Практична робота №2

### Особливості обробки табличних даних

#### Теоретичні відомості

**DataGridView** - стандартний GUI компонент для відображення та редагування таблиць.

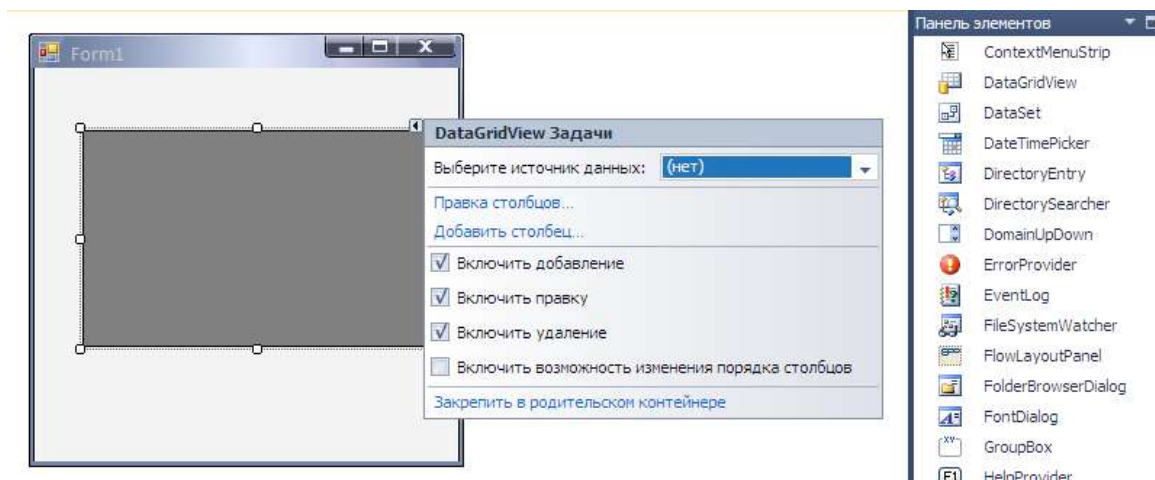


Джерело даних при створенні можна не вказувати, так як **DataGridView** дозволяє зберігати внутрішні дані, додавати чи знищувати їх під час виконання додатку. При налаштуванні властивостей компоненту **DataGridView** можна змінити властивості встановлені за замовчуванням, що встановить заборону на додавання та знищення рядків:

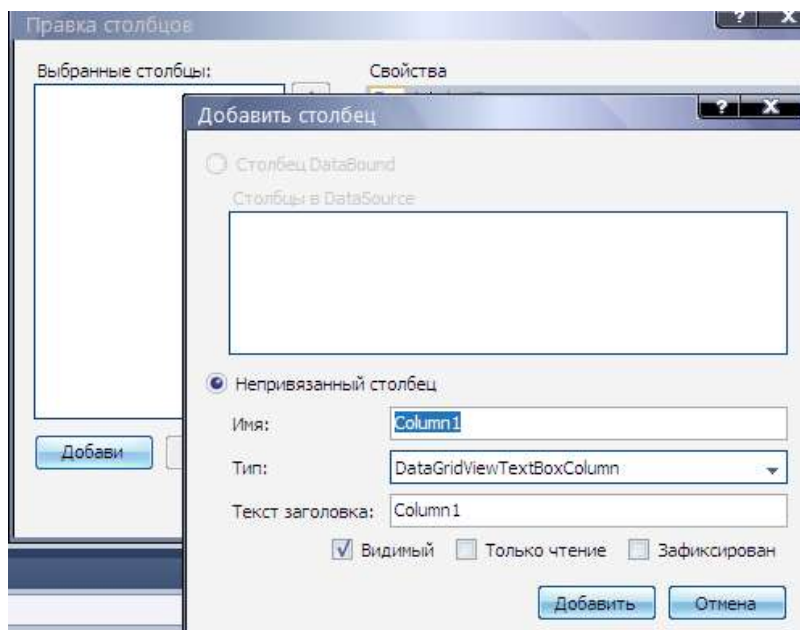
**AllowUserToAddRows = false,**

**AllowUserToDeleteRows = false,**

**ReadOnly = true.**



Стовпці та рядки можна додавати під час виконання програми. Якщо призначення та структура таблиці відома під час проектування додатку то стовпці додаються у вікні властивостей **Columns** (або **Правка столбцов**).



Звернення до комірок **DataGridView** аналогічне роботі з елементами масиву. Індексція стовпців (**Columns**) та рядків (**Rows**) починається з нуля. Стовпець з індексом 0 самий лівий, рядок - верхній. Кількість рядків визначається властивістю - **RowCount**.

Для додавання рядків використовують метод **Rows ->Add()**, для знищення – **Rows ->RemoveAt (номер рядка)**.

**Приклад:**

```
//додати у dataGridView1 - 4 рядки
dataGridView1->Rows->Add();
dataGridView1->Rows->Add();
dataGridView1->Rows->Add();
dataGridView1->Rows->Add();
//вилучити всі рядки з dataGridView1
while (0 != dataGridView1->RowCount)
dataGridView1->Rows->RemoveAt(0);
```

Змінювати дані у комірках таблиці можна лише тоді, коли існують відповідний стовпчик та рядок, у іншому випадку виникне помилка виконання. Після додавання рядка методом **Add** всі значення у комірках доданого рядка будуть порожніми. До комірки таблиці можна звертатися за номером стовпця и номером рядка через властивість **Value**.

### Приклад додавання та заповнення таблиці:

```
int i = 0;      //нумерація рядків починається з 0
if ( (dataGridView1->RowCount - 1) < i ) dataGridView1->Rows->Add();
//№ рядка додаємо в перший стовпець Cells[0]
dataGridView1->Rows[i]->Cells[0]->Value = (i+1).ToString();
//текст додаємо в другий стовпець Cells[1]
dataGridView1->Rows[i]->Cells[1]->Value = " текст";
i++; //переходимо до наступного рядка
dataGridView1->Rows[i]->Cells[0]->Value = (i+1).ToString();
```

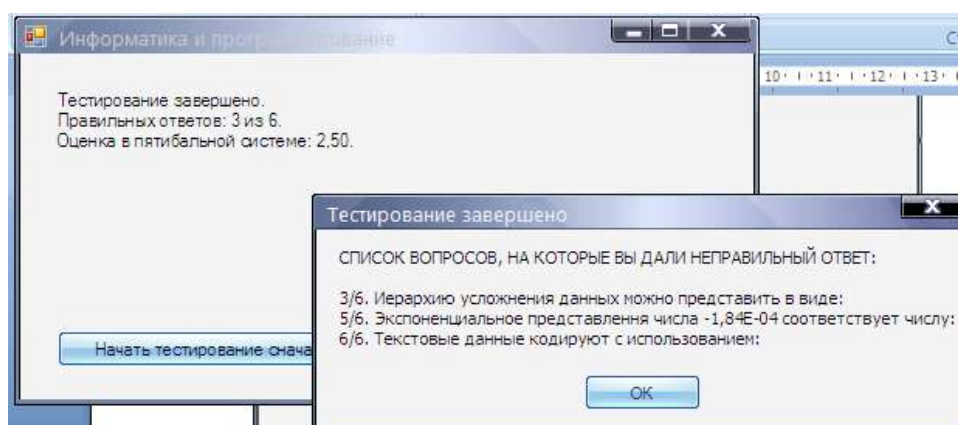
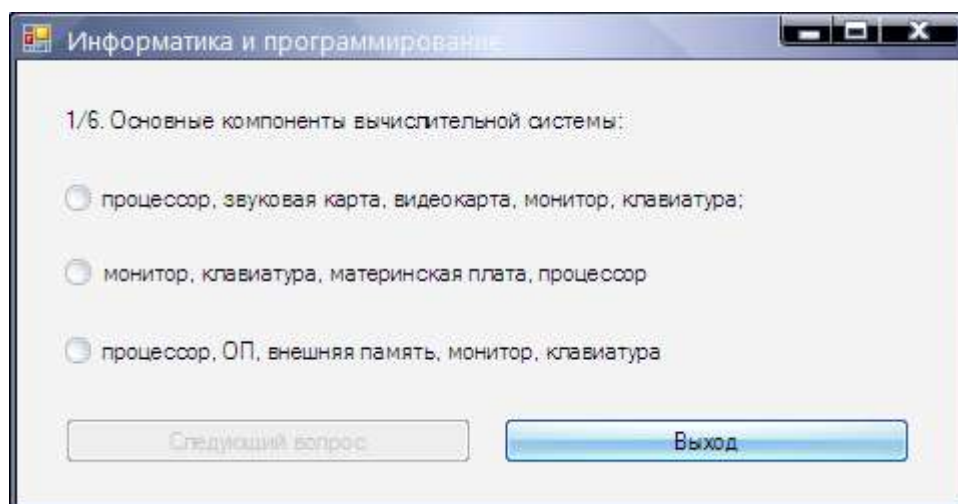
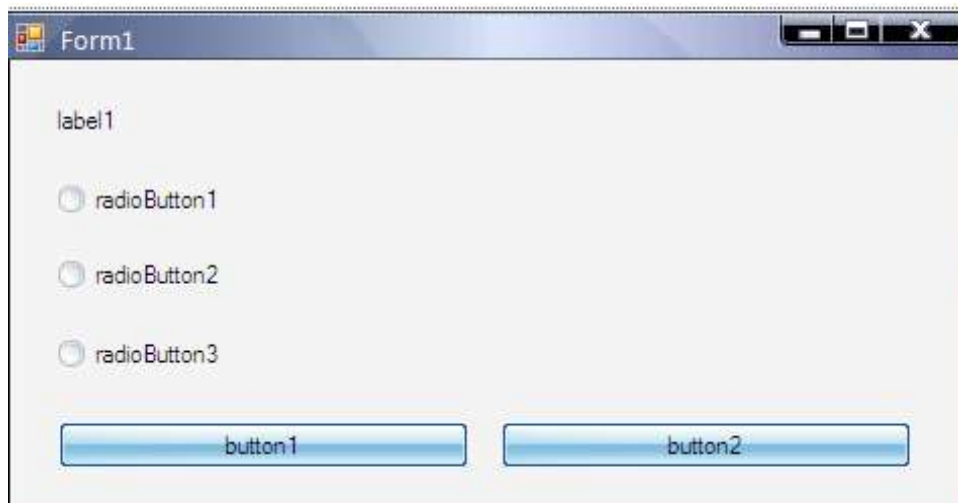
### Завдання для самостійного виконання

1. Замінити всі від'ємні елементи одновимірного масиву цілих чисел на значення найменшого з додатних чисел (нуль не враховувати).
2. Підрахувати кількість пар сусідніх елементів одновимірного масиву цілих чисел з однаковими значеннями.
3. Визначити, яке значення зустрічається у масиві дійсних чисел найчастіше.
4. Визначити суму елементів головної діагоналі.
5. Написати програму, яка заміняє всі елементи першого та останнього рядку та стовпчику на „0”.
6. Заповніть двовимірний масив випадковими числами. Відобразіть матрицю симетрично відносно головної діагоналі

# Практична робота №3

## Особливості розробки прикладних додатків

### Розробка тестової оболонки





**Завдання.** Проаналізувати представлений програмний код. Реалізувати тестову оболонку.

```
// ----- код створюється автоматично-----
```

```
#pragma once
```

```
namespace Тестирование {
```

```
using namespace System;
```

```
using namespace System::ComponentModel;
```

```
using namespace System::Collections;
```

```
using namespace System::Windows::Forms;
```

```
using namespace System::Data;
```

```
using namespace System::Drawing;
```

```
/// <summary>
```

```
/// Сводка для Form1
```

```
/// </summary>
```

```
public ref class Form1 : public System::Windows::Forms::Form
```

```
{
```

```
public:
```

```
Form1(void)
```

```
{
```

```
InitializeComponent();
```

```
//
```

```
//TODO: добавьте код конструктора
```

```
//
```

```
}
```

```
protected:
```

```
/// <summary>
```

```
/// Освободить все используемые ресурсы.
```

```
/// </summary>
```

```
~Form1()
```

```
{
```

```
if (components)
```

```
{
```

```
delete components;
```

```
}
```

```
}
```

```
private: System::Windows::Forms::Label^ label1;
```

```
protected:
```

```
private: System::Windows::Forms::Button^ button1;
```

```
private: System::Windows::Forms::Button^ button2;
```

```
private: System::Windows::Forms::RadioButton^ radioButton1;
```

```
private: System::Windows::Forms::RadioButton^ radioButton2;
```

```
private: System::Windows::Forms::RadioButton^ radioButton3;
```

```

private:
    /// <summary>
    /// Требуется переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Обязательный метод для поддержки конструктора - не изменяйте
    /// содержимое данного метода при помощи редактора кода.
    /// </summary>
    void InitializeComponent(void)
    {
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->button2 = (gcnew System::Windows::Forms::Button());
        this->radioButton1 = (gcnew System::Windows::Forms::RadioButton());
        this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());
        this->radioButton3 = (gcnew System::Windows::Forms::RadioButton());
        this->SuspendLayout();
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(21, 23);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(35, 13);
        this->label1->TabIndex = 0;
        this->label1->Text = L"label1";
        //
        // button1
        //
        this->button1->Location = System::Drawing::Point(24, 178);
        this->button1->Name = L"button1";
        this->button1->Size = System::Drawing::Size(203, 23);
        this->button1->TabIndex = 1;
        this->button1->Text = L"button1";
        this->button1->UseVisualStyleBackColor = true;
        this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
        //
        // button2
        //
        this->button2->Location = System::Drawing::Point(243, 178);
        this->button2->Name = L"button2";
    }

```

```

this->button2->Size = System::Drawing::Size(203, 23);
this->button2->TabIndex = 2;
this->button2->Text = L"button2";
this->button2->UseVisualStyleBackColor = true;
this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
//
// radioButton1
//
this->radioButton1->AutoSize = true;
this->radioButton1->Location = System::Drawing::Point(24, 60);
this->radioButton1->Name = L"radioButton1";
this->radioButton1->Size = System::Drawing::Size(85, 17);
this->radioButton1->TabIndex = 3;
this->radioButton1->TabStop = true;
this->radioButton1->Text = L"radioButton1";
this->radioButton1->UseVisualStyleBackColor = true;
//
// radioButton2
//
this->radioButton2->AutoSize = true;
this->radioButton2->Location = System::Drawing::Point(24, 97);
this->radioButton2->Name = L"radioButton2";
this->radioButton2->Size = System::Drawing::Size(85, 17);
this->radioButton2->TabIndex = 4;
this->radioButton2->TabStop = true;
this->radioButton2->Text = L"radioButton2";
this->radioButton2->UseVisualStyleBackColor = true;
//
// radioButton3
//
this->radioButton3->AutoSize = true;
this->radioButton3->Location = System::Drawing::Point(24, 136);
this->radioButton3->Name = L"radioButton3";
    this->radioButton3->Size = System::Drawing::Size(85, 17);
this->radioButton3->TabIndex = 5;
this->radioButton3->TabStop = true;
this->radioButton3->Text = L"radioButton3";
this->radioButton3->UseVisualStyleBackColor = true;
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(473, 222);
this->Controls->Add(this->radioButton3);

```

```

this->Controls->Add(this->radioButton2);
this->Controls->Add(this->radioButton1);
this->Controls->Add(this->button2);
this->Controls->Add(this->button1);
this->Controls->Add(this->label1);
this->Name = L"Form1";
this->Text = L"Form1";
this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
// ----- код створюється автоматично-----

```

```

this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
int СчетВопросов;
int ПравилОтветов;
int НеПравилОтветов;
array<String^>^ НеПравилОтветы;
int НомерПравОтвета;
int ВыбранОтвет;
IO::StreamReader^ Читатель;
private: System::
Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
button1->Text = "Наступне питання";
button2->Text = "Вихід";
radioButton1->CheckedChanged +=
gcnew EventHandler(this, &Form1::ИзмСостПерекл);
radioButton2->CheckedChanged +=
gcnew EventHandler(this, &Form1::ИзмСостПерекл);
radioButton3->CheckedChanged +=
gcnew EventHandler(this, &Form1::ИзмСостПерекл);
НачалоТеста();
}
void НачалоТеста()
{
System::Text::Encoding^ Кодировка =
System::Text::Encoding::GetEncoding(1251);
try
{
Читатель = gcnew IO::StreamReader(IO::Directory::GetCurrentDirectory()
+ "\\test.txt", Кодировка);
this->Text = Читатель->ReadLine();
СчетВопросов = 0; ПравилОтветов = 0; НеПравилОтветов = 0;
НеПравилОтветы = gcnew array<String^>(100);
}
}

```

```

catch (Exception^ Ситуация)
{
    MessageBox::Show(Ситуация->Message, "Помилка",
    MessageBoxButtons::ОК, MessageBoxIcon::Exclamation);
}
ЧитательСледВопрос();
}
void ЧитательСледВопрос()
{
    label1->Text = Читатель->ReadLine();
    radioButton1->Text = Читатель->ReadLine();
    radioButton2->Text = Читатель->ReadLine();
    radioButton3->Text = Читатель->ReadLine();
    НомерПравОтвета = int::Parse(Читатель->ReadLine());
    radioButton1->Checked = false; radioButton2->Checked = false;
    radioButton3->Checked = false;
    button1->Enabled = false;
    СчетВопросов = СчетВопросов + 1;
    if (Читатель->EndOfStream == true) button1->Text = "Завершить";
}
private: Void ИзмСостПереключ(System::Object^ sender, System::EventArgs^ e)
{
    button1->Enabled = true; button1->Focus();
    RadioButton^ Переключатель = (RadioButton^)sender;
    String^ tmp = Переключатель->Name;
    ВыбранОтвет = int::Parse(tmp->Substring(11));
}
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (ВыбранОтвет == НомерПравОтвета)
        ПравилОтветов = ПравилОтветов + 1;
    if (ВыбранОтвет != НомерПравОтвета)
    {
        НеПравилОтветов = НеПравилОтветов + 1;
        НеПравилОтветы[НеПравилОтветов] = label1->Text;
    }
    if (button1->Text == "Почти тест")
    {
        button1->Text = "Наступне питання";

        radioButton1->Visible = true; radioButton2->Visible = true;
        radioButton3->Visible = true;
        НачалоТеста(); return;
    }
    if (button1->Text == "Завершити")

```

```

{
Читатель->Close();
radioButton1->Visible = false; radioButton2->Visible = false;
radioButton3->Visible = false;
label1->Text = String::Format("Тестування завершено.\n" +
"Вірних відповідей: {0} из {1}.\n" +
"Оцінка: {2:F2}.", ПравилОтветов,
СчетВопросов, (ПравилОтветов * 5.0F) / СчетВопросов);
button1->Text = "Розпочати тестування";
String^ Str = "СПИСОК ПИТАНЬ, НА ЯКІ ВИ ДАЛИ " + "НЕВІРНІ
ВІДПОВІДІ:\n\n";
for (int i = 1; i <= НеПравилОтветов; i++)
Str = Str + НеПравилОтветы[i] + "\n";
if (НеПравилОтветов != 0)
MessageBox::Show(Str, "Тестування завершено");
}
if (button1->Text == "Наступне питання") ЧитатьСледВопрос();}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{ this->Close();};};

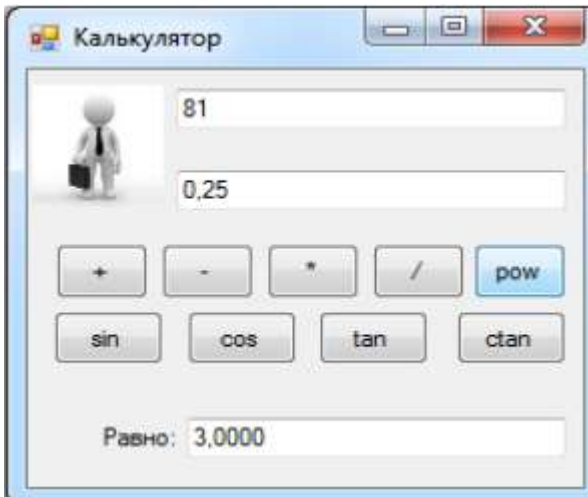
```

## Практична робота №4

### Ініціювання обробки події клавіатури



#### Розробка прикладного додатку «Калькулятор»



```
#pragma endregion
```

```
String^ TorZ; // Крапка або кома
```

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
```

```
{ this->Text = " Тільки цифри";
```

```
label1->Text = " Можна вводити тільки цифри!";
```

```
// З'ясуємо що встановлено в настройках як роздільник
```

```
// крапка або кома
```

```
TorZ = Globalization::NumberFormatInfo::CurrentInfo->NumberDecimalSeparator; }
```

```
private: System::Void textBox1_KeyPress(System::Object^ sender,
```

```
System::Windows::Forms::KeyPressEventArgs^ e)
```

```
{
```

```
bool TZFound = false; // Розділовий знак знайдений
```

```
if (Char::IsDigit(e->KeyChar) == true) return;
```

```
if (e->KeyChar == (char)Keys::Back) return;
```

```
if (textBox1->Text->IndexOf(TorZ) != -1)
```

```
TZFound = true;
```

```
if (TZFound == true) { e->Handled = true; return; }
```

```
if (e->KeyChar.ToString() == TorZ) return;
```

```
e->Handled = true; }
```

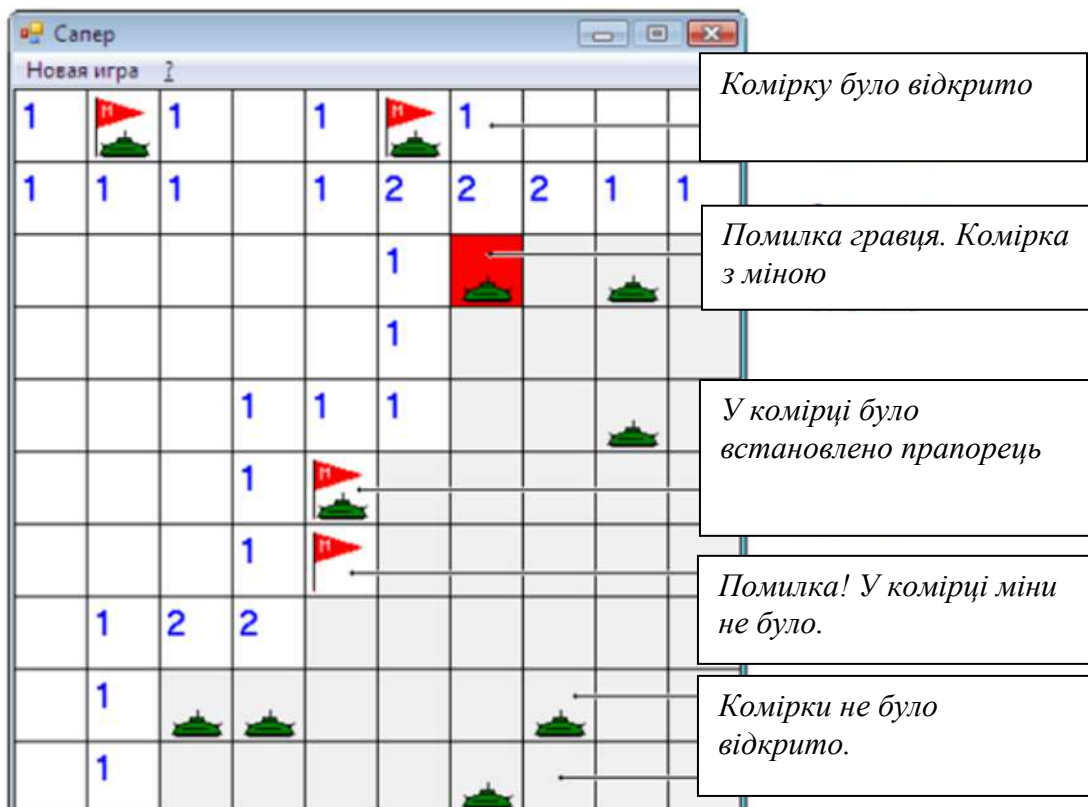
## Практична робота №5

### Особливості роботи з графічними даними

**Завдання.** Розробити ігровий додаток «Сапер».

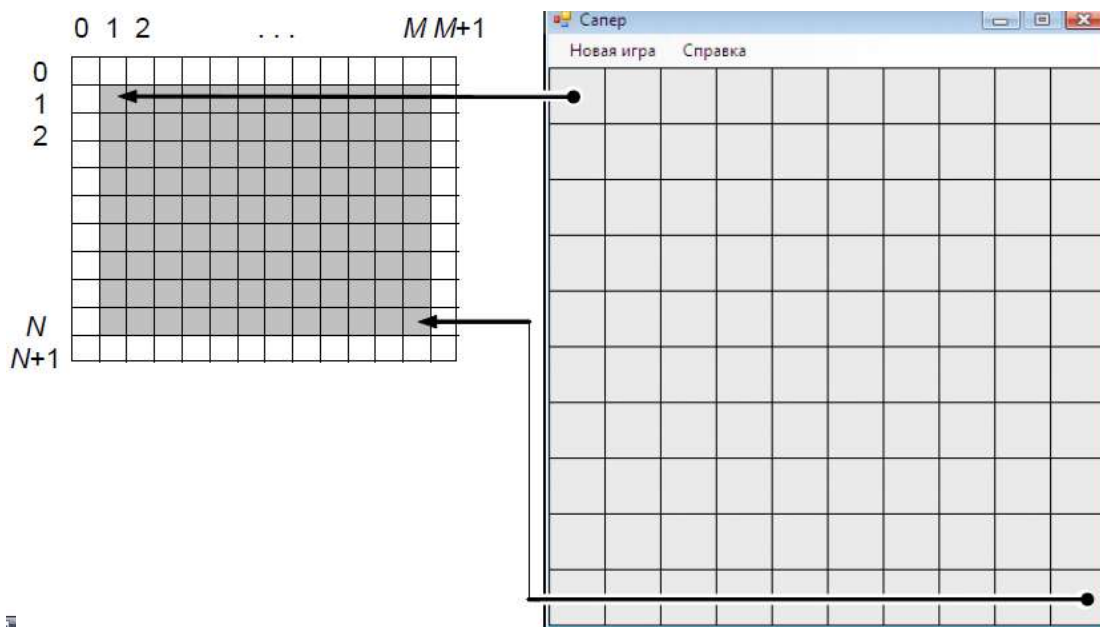
Ігрове поле складається з кліток, в кожній з яких може бути міна. Задача гравця - знайти всі міни і помітити їх прапорцями.

Використовуючи кнопки миші, гравець може відкрити комірку або поставити в неї прапорець, вказавши тим самим, що в комірці знаходиться міна. Комірка відкривається клацанням лівої кнопки миші, прапорець ставиться клацанням правої. Якщо в комірці, яку відкрив гравець, є міна, то відбувається вибух і гра закінчується. Якщо у комірці міни немає, то в ній з'являється число, відповідне кількості мін, що знаходяться в сусідніх комірках. Аналізуючи інформацію про кількість мін в комірках, сусідніх з уже відкритими, гравець може помітити прапорцями всі міни. Обмежень на кількість комірок, помічених прапорцями немає. Проте для завершення гри (виграшу) прапорці повинні бути встановлені тільки в тих комірках, в яких є міни. Помилково встановлений прапорець можна прибрати, клацнувши правою кнопкою миші в комірці, в якій він знаходиться.





У програмі ігрове поле представлено масивом  $N + 2$  на  $M + 2$ , де  $N M$  - розмір ігрового поля. Елементи масиву з номерами рядків від 1 до  $N$  і номерів стовпців від 1 до  $M$  відповідають коміркам ігрового поля, перші і останні стовпці і рядки відповідають межі ігрового поля.



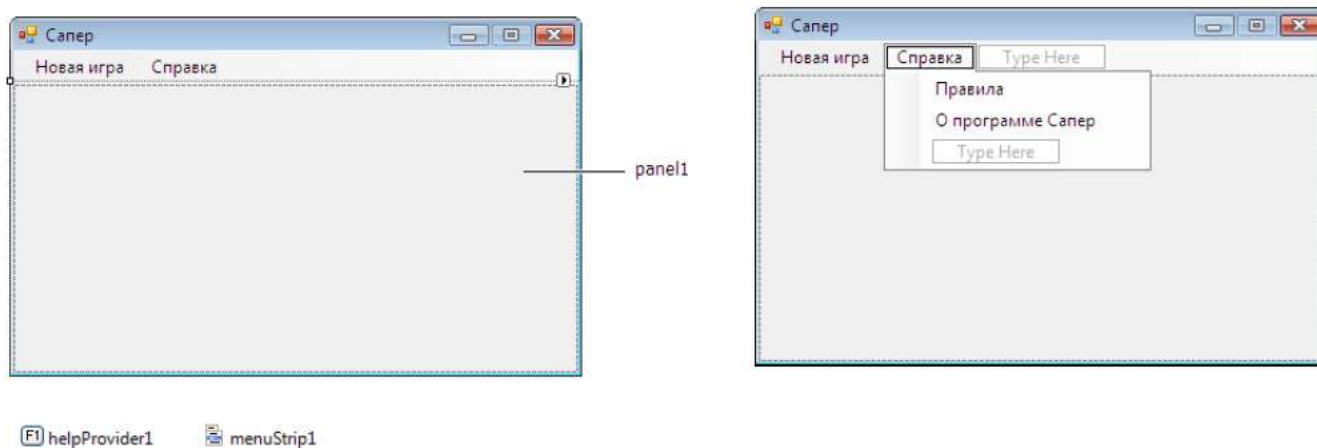
На початку гри кожен елемент масиву, відповідний коміркам ігрового поля, може містити число від 0 до 9. Нуль відповідає порожній комірці, поряд з якою немає мін. Коміркам, в яких немає мін, але поряд з якими міні є, відповідають числа від 1 до 8. Елементи масиву (комірки) в яких знаходяться міні, мають значення 9.

Елементи масиву, відповідні межі поля, містять -3.

-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-3	9	1	0	0	0	0	0	0	0	0	-3
-3	1	1	0	0	0	0	0	0	0	0	-3
-3	1	2	2	1	0	0	0	1	1	1	-3
-3	1	9	9	1	0	0	0	2	9	2	-3
-3	1	2	2	1	0	0	0	2	9	3	-3
-3	0	0	0	0	0	0	0	2	3	9	-3
-3	0	1	2	2	1	0	0	1	9	2	-3
-3	0	2	9	9	1	0	0	1	1	1	-3
-3	0	2	9	3	1	0	0	0	0	0	-3
-3	0	1	1	1	0	0	0	0	0	0	-3
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3

В процесі гри стан ігрового поля змінюється (гравець відкриває комірки і ставить прапорці), і відповідно змінюються значення елементів масиву. Якщо гравець поставив у комірку прапорець, то значення відповідного елементу масиву збільшується на 100. Наприклад, якщо прапорець поставлений вірно - у клітку, в якій є міна, то значення відповідного елементу масиву стане 109. Якщо прапорець поставлений помилково, наприклад в порожню комірку, елемент масиву міститиме число 100. Якщо гравець відкриє комірку, то значення елементу масиву збільшується на 200. Такий спосіб кодування дозволяє зберегти інформацію про початковий стан комірки.

## Крок 1. Форма



Text	Сапер
FormBorderStyle	FixedSingle
MaximizeBox	False
StartPosition	CenterScreen

Для компоненту **panel1** властивість **Dock = Fill** (розміщення компоненту на всю форму)

## Крок 2. Модуль головної форми

**private:**

**static int**

MR = 10, // к-ть комірок по-вертикалі

MC = 10, // к-ть комірок по-горизонталі

NM = 10, // к-ть мін

W = 40, // ширина комірки

H = 40; // висота комірки

// ігрове поле

static array<int,2>^ Pole = gnewarray<int,2>(MR + 2, MC + 2);

```

intnMin; // к-ть знайдених мін
intnFlag; // к-ть встановлених прапорців
// статус гри
intstatus;
// 0 — початок гри,
// 1 — гра,
// 2 — результат
// графічна поверхня форми
System::Drawing::Graphics^ g;

// конструктор
public:
Form1(void)
{
InitializeComponent();
for(introw = 0; row<= MR+1; row++)
{
Pole[row,0] = -3;
Pole[row,MC+1] = -3;
}

for(intcol = 0; col<= MC+1; col++)
{
Pole[0,col] = -3;
Pole[MR+1,col] = -3;
}
// встановлення розміру форми
this->ClientSize = System::Drawing::Size(W*MC + 1,
H*MR + menuStrip1->Height + 1);
newGame(); // нова гра
// графічна поверхня
g = panel1->CreateGraphics();
}

```

### Крок 3. Нова гра. Побудова поля з комірок.

```

voidnewGame()
{
introw, col; //
intn = 0; // кількість поставлених мін
intk; // кількість мін у сусідніх комірках
// очистити поле
for(row = 1; row<= MR; row++)
for(col = 1; col<= MC; col++)
Pole[row,col] = 0;
Random^ rnd = gcnewRandom();
// встановлення мін
do

```

```

{
row = rnd->Next(MR) + 1;
col = rnd->Next(MC) + 1;
if(Pole[row,col] != 9)
{
Pole[row,col] = 9;
n++;
}
}
while(n != NM);

// для кожної комірки обчислюємо к-ть мін у сусідніх комірках
for(row = 1; row<= MR; row++)
for(col = 1; col<= MC; col++)
if(Pole[row,col] != 9)
{
k = 0;
if(Pole[row-1,col-1] == 9) k++;
if(Pole[row-1,col] == 9) k++;
if(Pole[row-1,col+1] == 9) k++;
if(Pole[row,col-1] == 9) k++;
if(Pole[row,col+1] == 9) k++;
if(Pole[row+1,col-1] == 9) k++;
if(Pole[row+1,col] == 9) k++;
if(Pole[row+1,col+1] == 9) k++;

Pole[row,col] = k;
}
status = 0; // початок гри
nMin = 0; // міни не знайдено
nFlag = 0; // прапорці не встановлено
}

// малювання поля
voidshowPole(Graphics^ g, intstatus)
{
for(introw = 1; row<= MR; row++)
for(intcol = 1; col<= MC; col++)
this->Kletka(g, row, col, status);
}

// малювання комірки
voidKletka(Graphics^ g, introw, intcol, intstatus)
{
intx, y; // координати лівого верхнього кута комірки
x = (col — 1) * W + 1;
y = (row-1)* H + 1;
// комірки, що не відкрили - сірі
if(Pole[row,col] < 100)

```

```

g->FillRectangle(SystemBrushes::ControlLight,
x-1, y-1, W, H);
// відкриті чи відмічені комірки
if(Pole[row,col] >= 100) {
// відкриті комірки - білі
if(Pole[row,col] != 109)
g->FillRectangle(Brushes::White, x-1, y-1, W, H);

else
// підбив міни
g->FillRectangle(Brushes::Red, x-1, y-1, W, H);
// кількість мін у сусідніх комірках
if((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
g->DrawString((Pole[row,col]-100).ToString(),
gcnewSystem::Drawing::Font("Tahoma", 10, System::Drawing::FontStyle::Regular),
Brushes::Blue, (float)x+3, (float)y+2);
}

// встановлення прапора
if(Pole[row,col] >= 200)
this->flag(g, x, y);
// межі комірки
g->DrawRectangle(Pens::Black, x-1, y-1, W, H);
// завершення гри (status = 2), відображення мін
if((status == 2) && ((Pole[row,col] % 10) == 9))
this->mina(g, x, y);
}

```

#### Крок 4. Опис гри

```

// клік по комірці ігрового поля
private: System::Void panel1_MouseClick(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{
if(status == 2)
// гра завершена
return;

if(status == 0)
// перший клік
status = 1;
//перетворення координат миші в індекси комірки поля
// (e.X, e.Y) — координати точки форми,
introw, col;
row = e->Y/H + 1;
col = e->X/W + 1;
// координати області виведення
intx = (col-1)* W + 1,
y = (row-1)* H + 1;

```

```

// клік ЛКМ
if(e->Button == System::Windows::Forms::MouseButtons::Left)
{
// відкрита комірка з міною
if(Pole[row,col] == 9)
{
Pole[row,col] += 100;
// гру завершено
status = 2;
// перемальовування форми
this->panel1->Invalidate();
}
Else
if(Pole[row,col] < 9)
// відкрити комірку
this->open(row,col);
}
// клік ПКМ
if(e->Button == System::Windows::Forms::MouseButtons::Right)
{
// ставим прапорець у комірку
if(Pole[row,col] <= 9) {
nFlag += 1;

if(Pole[row,col] == 9)
nMin += 1;
Pole[row,col] += 200;
if((nMin == NM) && (nFlag == NM)) {
// гру завершено
status = 2;
// перемальовування ігрового поля
this->Invalidate();
}
else
// перемальовування комірки
this->Kletka(g, row, col, status);
}
else
// прибираємо прапор
if(Pole[row,col] >= 200)
{
nFlag -= 1;
Pole[row,col] -= 200;
// перемальовуємо комірку
this->Kletka(g, row, col, status);
}
}

```

```
}  
}
```

Функція **flag** малює прапор

Функція **mina** малює міну

## Крок 5.

*// обробка події Paint*

```
private: System::Void panel1_Paint(System::Object^ sender,  
System::Windows::Forms::PaintEventArgs^ e)  
{ showPole(g, status); }
```

*// відкриваємо комірки у яких немає мін*

```
void open(int row, int col)
```

```
{
```

```
int x = (col-1)* W + 1,
```

```
y = (row-1)* H + 1;
```

```
if (Pole[row,col] == 0)
```

```
{
```

```
Pole[row,col] = 100;
```

```
// відображення вмісту комірок
```

```
this->Kletka(g, row, col, status);
```

```
this->open(row, col-1);
```

```
this->open(row-1, col);
```

```
this->open(row, col+1);
```

```
this->open(row+1, col);
```

```
this->open(row-1,col-1);
```

```
this->open(row-1,col+1);
```

```
this->open(row+1,col-1);
```

```
this->open(row+1,col+1);
```

```
}
```

```
else
```

```
if ((Pole[row,col] < 100) &&(Pole[row,col] != -3))
```

```
{
```

```
Pole[row,col] += 100;
```

```
this->Kletka(g, row, col, status);
```

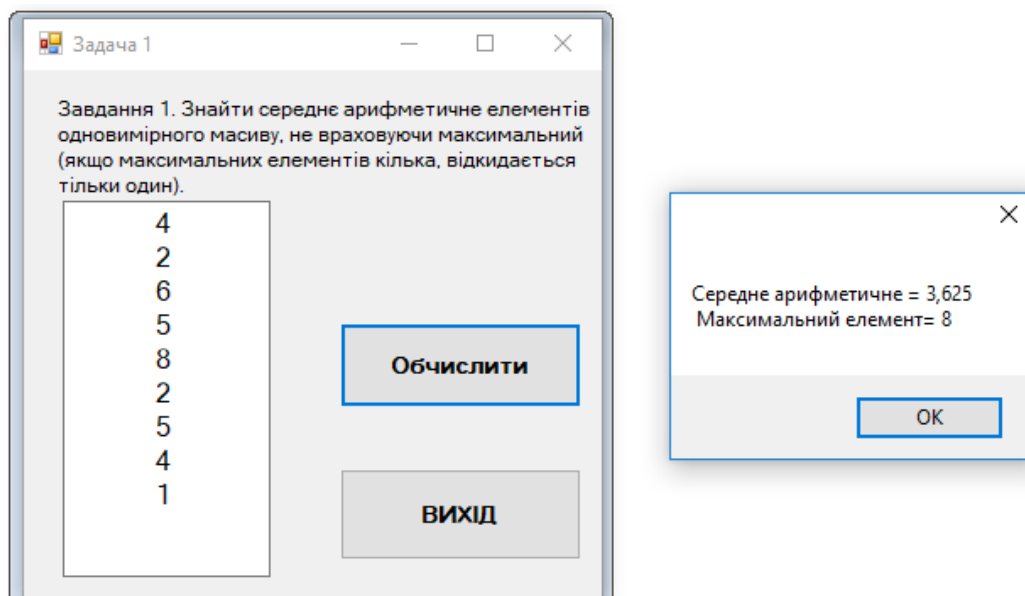
```
}
```

```
}
```

## ДОДАТКИ

### Додаток А

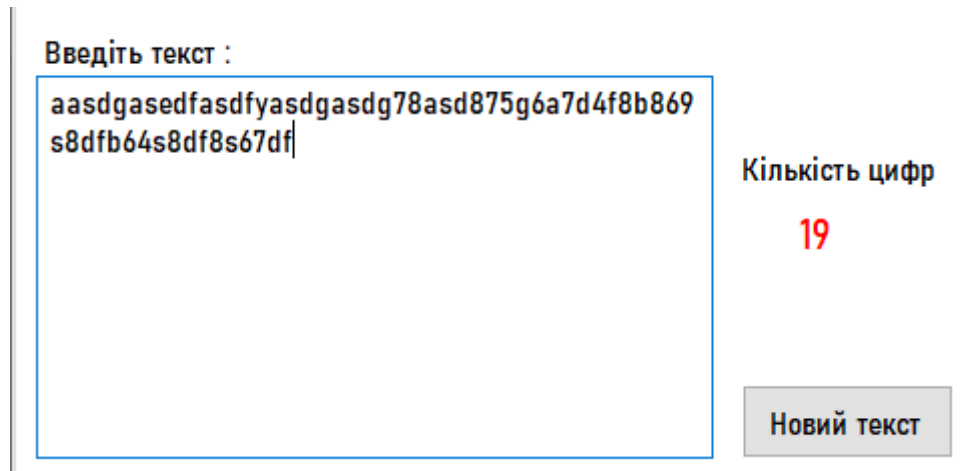
**Приклад програми** Знайти середнє арифметичне елементів одновимірного масиву, не враховуючи максимальний (якщо максимальних елементів кілька, відкидається тільки один).



```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int a[9],i,f=0;
    double sa, s=0,max;//оголошення змінних
    textBox1->Clear();//очищення TextBox
    Random^r = gcnew Random;//генерація елементів масиву
    // Заповнення масиву та знаходження суми
    for (i = 0; i < 9; i++)//цикл з параметром
    {
        a[i] = r->Next(10);// надання змінній випадкового значення
        textBox1->AppendText(a[i] + " \r\n");//виведення елементів масиву у TextBox
        s+=a[i]; //знаходження суми елементів масиву
    }
    //Знаходження максимального елементу масиву
    max=a[0]; // вважаємо максимальним - перший елемент масиву
    for (i=0; i<9; i++)//цикл з параметром
    {
        if (a[i]>max) max =a[i];//якщо знайдеться елемент > max присвоїти нове
значення
    }
    // Знаходження середнього значення (не враховуючи максимальний)
    sa=(s- max)/8;
    // Виведення результату
    MessageBox::Show("Середнє арифметичне = "+sa+"\r\n Максимальний
елемент= "+ max);
}
```



**Приклад програми** Підрахувати кількість чисел у введеному з клавіатури тексті. Програма передбачає обробку помилок (виключення).



Введіть текст :

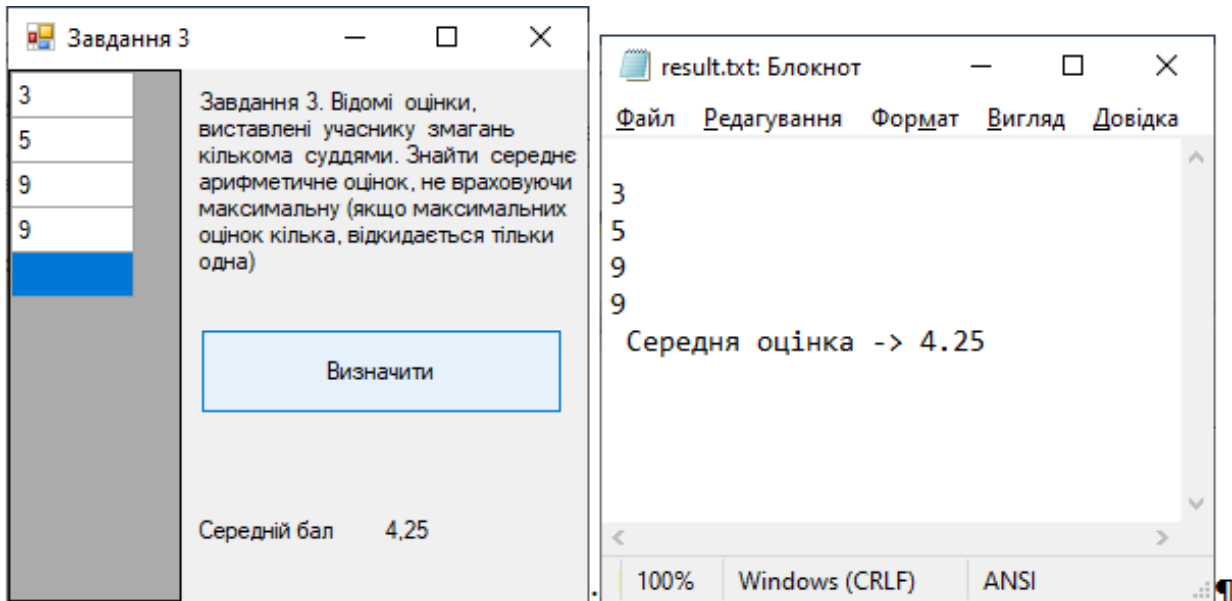
aasdgasdfasdfyasdgasdg78asd875g6a7d4f8b869  
s8dfb64s8df8s67df

Кількість цифр  
**19**

Новий текст

```
private: System::Void textBox1_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e)
{
    try
    {
        int k;
        k=Convert::ToInt32(label3->Text);
        if(e->KeyChar>='0' && e->KeyChar<='9') k++;
        label3->Text=Convert::ToString(k);
    }
    catch (...)
    {
        MessageBox::Show("Помилка. Можливо некоректно введені дані.", "Error",
        MessageBoxButtons::OK, MessageBoxIcon::Error);
    }
}
```

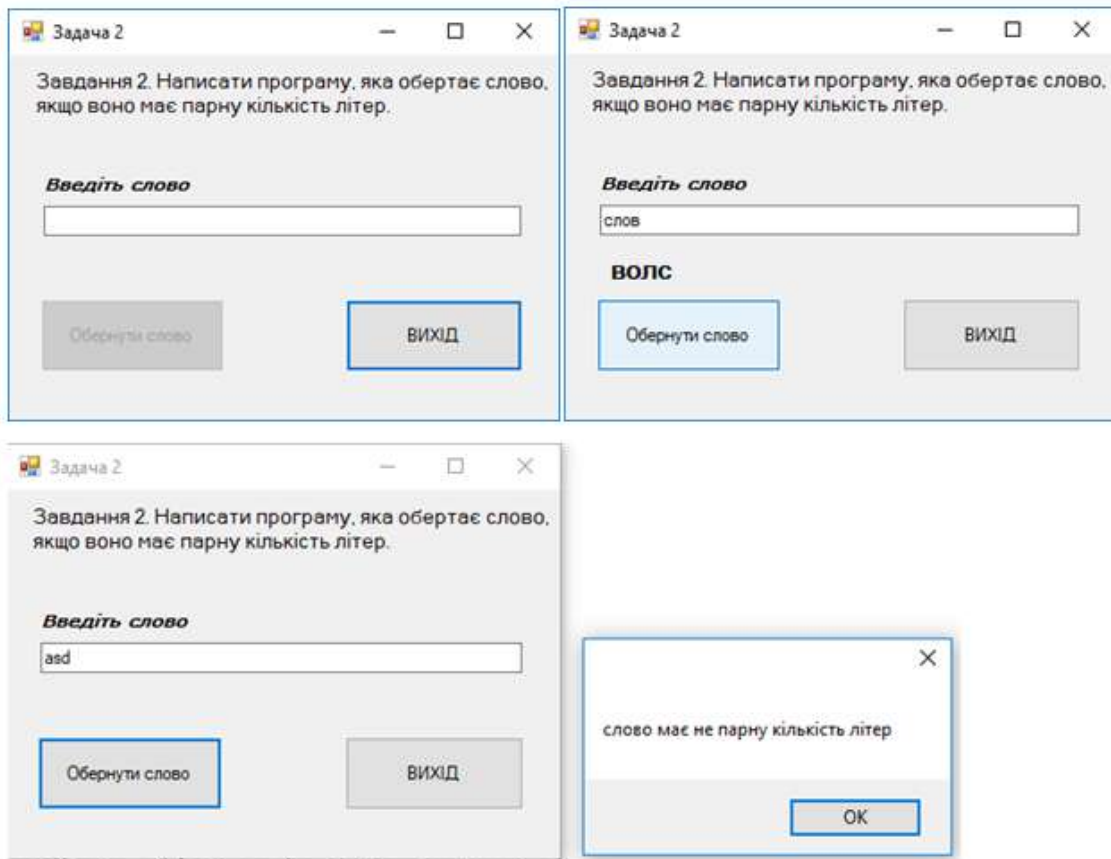
**Приклад програми** 3 використанням компоненту **dataGridView** та записом результату у файл. Програма передбачає обробку помилок (виключення).



```
#include <fstream>
....

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    try
    {
        std::ofstream file1;
        file1.open("C:\\1\\result.txt");
        int n=(dataGridView1->RowCount)-1;
        double max=0;
        double sum=0;
        double ser;
        double *A = new double [n];
        for(int i=0;i<n;i++)
        {
            A[i]=Convert::ToDouble(dataGridView1->Rows[i]->Cells[0]->Value);
            if(A[i]>max) max=A[i];
            sum=sum+A[i];
            file1<<"\n"<<A[i];
        }
        sum=sum-max;
        ser=sum/n;
        file1<<"\n Середня оцінка -> "<<ser;
        label3->Text=Convert::ToString(ser);
    }
    catch (...)
    {
        MessageBox::Show("Помилка. Можливо некоректно введені дані. Використовуйте числові символи", "Error", MessageBoxButtons::OK, MessageBoxIcon::Error);
    }
}
```

**Приклад програми** Написати програму, яка обертає слово, якщо воно має парну кількість літер.

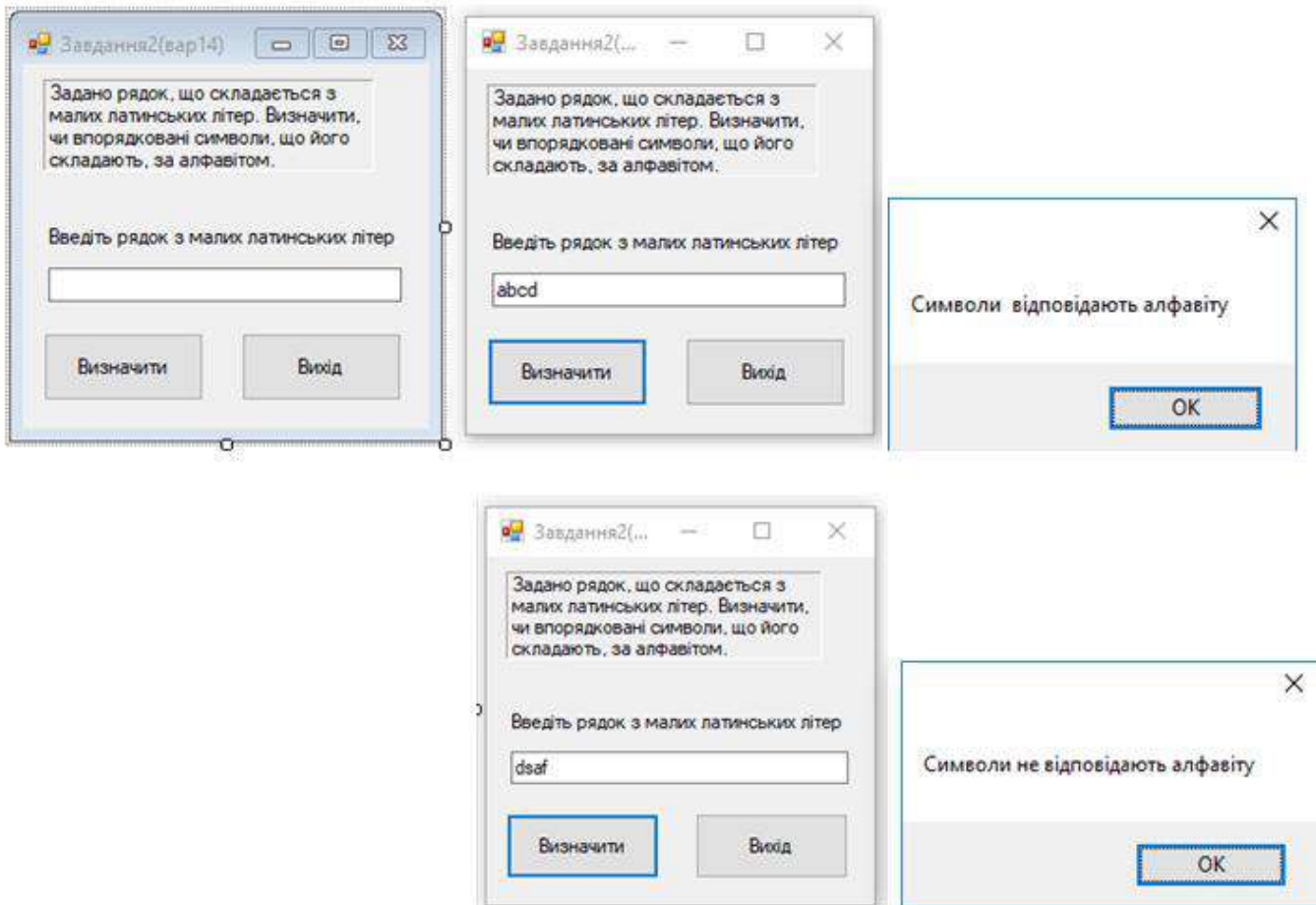


```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    Close(); // вихід із програми
}
```

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    String^ s; //оголошення рядкової змінної
    String^ s1 = "";
    s=textBox1->Text;
    int k= s->Length;// визначення довжини слова
    if (k%2==0) // якщо слово має парну к-ть літер інвертуємо його
    {
        for(int i=k-1;i>=0;i--)s1+=s[i];
        label3->Text=s1; // виведення інвертованого слова
    }
    else MessageBox::Show("слово має не парну кількість літер");
}
```

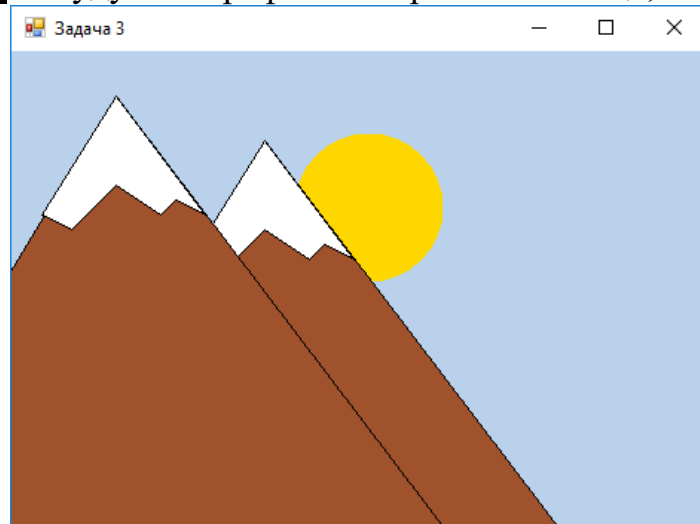
```
private: System::Void textBox1_TextChanged( )
{
    //коли рядок введено, кнопка стане активною
    if(textBox1->Text!="")button2->Enabled=true;
}
```

**Приклад програми** . Задано рядок, що складається з малих латинських літер. Визначити, чи впорядковані символи, що його складають, за алфавітом.



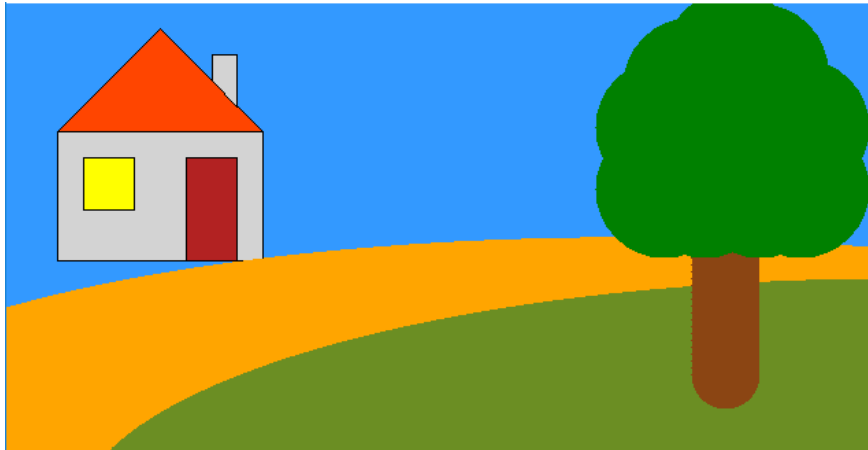
```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    Close(); // Вихід
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    // рядок вводиться з клавіатури
    String^ s = textBox1->Text;
    // рядок впорядкованих символів
    String^ s1 = "abcdefghijklmnopqrstuvwxyz";
    // метод визначення входження рядка s в рядок s1
    bool b = s1->Contains(s);
    // якщо підрядок входить в рядок то b=true, це означає, що символи
    // впорядковані за алфавітом
    if(b) MessageBox::Show("Символи відповідають алфавіту");
    else MessageBox::Show("Символи не відповідають алфавіту");
}
```

Приклад програми. Побудувати графічне зображення сонця, яке заходить за скелю.



```
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
    e->Graphics->FillEllipse(Brushes::Gold,190,55,100,100);// сонце
    int h=-150,u=50;
    for(int i=0;i<2;i++)
    {
        array<Point>^ t1;
        t1 = gcnew array<Point>(3); // масив точок для побудови скелі
        t1[0].X = 320+h; t1[0].Y = 10 +u;
        t1[1].X = 146+h; t1[1].Y = 300+u;
        t1[2].X = 540+h; t1[2].Y = 300+u;
        //мальовання скель
        e->Graphics->FillPolygon(Brushes::Sienna, t1);
        e->Graphics->DrawPolygon(Pens::Black,t1);
        array<Point>^ p1;
        p1 = gcnew array<Point>(9); // масив точок для побудови шапки
        p1[0].X = 320+h; p1[0].Y = 10+u ;
        p1[1].X = 380+h; p1[1].Y = 90+u ;
        p1[2].X = 360+h; p1[2].Y = 80+u ;
        p1[3].X = 350+h; p1[3].Y = 90+u ;
        p1[4].X = 320+h; p1[4].Y = 70+u ;
        p1[5].X = 310+h; p1[5].Y = 80+u ;
        p1[6].X = 300+h; p1[6].Y = 90+u ;
        p1[7].X = 290+h; p1[7].Y = 100+u ;
        p1[8].X = 270+h; p1[8].Y = 90+u ;
        //снігова шапка
        e->Graphics->FillPolygon(Brushes::White, p1);
        e->Graphics->DrawPolygon(Pens::Black,p1);
        h=-250,u=20;
    }
}
```

Приклад програми. Побудувати графічне зображення будиночка



```
private: System::Void pictureBox1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
    double x=0.75;
    //дах будинка
    array<Point>^p=gcnew array<Point>(3);
    p[0].X=x*50;p[0].Y=x*125;
    p[1].X=x*250;p[1].Y=x*125;
    p[2].X=x*150;p[2].Y=x*25;
    e->Graphics->FillPolygon(Brushes::OrangeRed,p);
    e->Graphics->DrawPolygon(Pens::Black,p);
    //будинок
    array<Point>^r=gcnew array<Point>(4);
    r[0].X=50*x;r[0].Y=125*x;
    r[1].X=250*x;r[1].Y=125*x;
    r[2].X=250*x;r[2].Y=250*x;
    r[3].X=50*x;r[3].Y=250*x;
    e->Graphics->FillPolygon(Brushes::LightGray ,r);
    e->Graphics->DrawPolygon(Pens::Black,r);
    // Вікно
    array<Point>^k=gcnew array<Point>(4);
    k[0].X=75*x;k[0].Y=150*x;
    k[1].X=125*x;k[1].Y=150*x;
    k[2].X=125*x;k[2].Y=200*x;
    k[3].X=75*x;k[3].Y=200*x;
    e->Graphics->FillPolygon(Brushes::Yellow,k);
    e->Graphics->DrawPolygon(Pens::Black,k);
    //Димар
    array<Point>^z=gcnew array<Point>(4);
    z[0].X=200*x;z[0].Y=50*x;
    z[1].X=200*x;z[1].Y=75*x;
    z[2].X=225*x;z[2].Y=100*x;
    z[3].X=225*x;z[3].Y=50*x;
    e->Graphics->FillPolygon(Brushes::LightGray ,z);
    e->Graphics->DrawPolygon(Pens::Black,z);
}
```

```

//Дорога та галявина
array<Point>^ t=gcnew array<Point>(4);
t[0].X=175*x;t[0].Y=250*x;
t[1].X=225*x;t[1].Y=250*x;
t[2].X=225*x;t[2].Y=150*x;
t[3].X=175*x;t[3].Y=150*x;
e->Graphics->FillPolygon(Brushes::Firebrick,t);
e->Graphics->DrawPolygon(Pens::Black,t);
e->Graphics->FillEllipse(Brushes::Orange,-150,170,1200,300);
e->Graphics->FillEllipse(Brushes::OliveDrab,50,200,1200,350);
//Дерево
for(int x=30;x<250;x=x+5)
e->Graphics->FillEllipse(Brushes::SaddleBrown,500,x,50,50);
//Крона дерева
e->Graphics->FillEllipse(Brushes::Green,80+400,10-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,100+400,20-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,100+400,30-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,50+400,30-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,130+400,60-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,30+400,60-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,130+400,105-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,30+400,105-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,100+400,105-20,100,100);
e->Graphics->FillEllipse(Brushes::Green,60+400,105-20,100,100);
}

```

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вирт Н. Алгоритмы и структуры данных / Н.Вирт -- М.: Мир,1989. – 360 с.
2. Войтенко В.В. С/С++: Теорія та практика. Навчально-методичний посібник / В.В. Войтенко, А.В. Морозов – Житомир:ЖДТУ, 2004. – 324с.
3. Галузинський Г. П. Сучасні технологічні засоби обробки інформації: Навчальний посібник / Г.П.Галузинський, І.В.Гордієнко — К.: КНЕУ, 2008. — 224 с.
4. Глушаков С.В. Практикум по С++/ С.В. Глушаков, А.В. Коваль, С.В. Смиронов,– Харьков: Фолио, 2006. – 525 с.
5. Златопольский Д.М. Сборник задач по программированию /Д.М. Златопольский– 2-е изд., перераб. и доп. – СПб.:БХВ – Петербург, 2007. – 240с.
6. Златопольский, Д. М. Я иду на урок информатики: Задачи по программированию 7-11 классы [Текст] / Д. М. Златопольский. – М. : Первое сентября, 2001. – 208 с.
7. Кнут Д. Искусство программирования. Т. 2. Получисленные алгоритмы / Д.Кнут. – М.: Вильямс, 2007. — 832 с.
8. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск / Д.Кнут. – М.: Вильямс, 2007. — 824с.
9. Кривцова О.П. Информатика. Основи програмування у середовищі Microsoft Visual C++Express [Електронний ресурс] : навч. посіб. / О.П. Кривцова. – Полтава: ПНПУ імені В.Г. Короленка, 2018. – 161с.
- 10.Кривцова О.П. Програмування мовою С++. Розробка консольних додатків у середовищі Microsoft Visual C++Express [Електронний ресурс] : навч. посіб. / О.П. Кривцова. – Полтава: ПНПУ імені В.Г. Короленка, 2017. – 99с.
- 11.Культин Н. Б. С/С++ в примерах и задачах /Н.Б. Культин - СПб.: БХВ-Петербург, 2001. - 288с.
- 12.Культин Н. Б. Основы программирования в Microsoft Visual C++ 2010/Н.Б. Культин - СПб.: БХВ-Петербург, 2010. - 384с.
- 13.Пахомов Б.И. С.С++ и VC++2010 для начинающих /Б.И. Пахомов.- СПб.: БХВ-Петербург, 2011. - 736с.
- 14.Технологія програмування в історичному аспекті – Режим доступу: URL: [https://studopedia.su/2\\_27596\\_tehnologiya-programuvannya-v-istorichnomu-aspekti.html](https://studopedia.su/2_27596_tehnologiya-programuvannya-v-istorichnomu-aspekti.html)
- 15.Хортон А. VC++2010:полный курс /Айвор Хортон. – М.: ООО «И.Д. Вильямс», 2011. – 1216с.