Уманський національний університет садівництва

Факультет економіки та підприємництва

Кафедра інформаційних технологій

Алгоритмізація і програмування

Методичний посібник

та завдання ***для проходження навчальної практики*** студентами освітньо-кваліфікаційного рівня бакалавр, що навчаються за спеціальністю 122 "Комп'ютерні науки"

Умань – 2019

УДК 519.85

Рецензенти:

Дубовой В.В., доктор технічних наук, професор,
завідувач кафедри комп’ютерних систем управління
Вінницький національний технічний університет

Ковальов Л.Є., кандидат фізико-математичних наук, доцент
Уманський національний університет садівництва

Алгоритмізація і програмування: Методичний посібник та завдання для проходження навчальної практики студентами освітньо-кваліфікаційного рівня бакалавр, що навчаються за спеціальністю 122 "Комп'ютерні науки" / Гринчак О.В., Давлетханова О.Х., Марін Б.М.. – Умань: УНУС, 2019. – 32 с.

Схвалено і рекомендовано до друку кафедрою інформаційних технологій (протокол № ___ від ___ _____ 20____ року) та методичною комісією факультету економіки та підприємництва (протокол № __ від ___ _____ 20__ року)

Зміст

Пояснювальна записка

Програма навчальної практики студентів спеціальності "Комп'ютерні науки" розроблена відповідно до навчального плану дисципліни та Положення про проведення практики студентів Уманського НУС.

На початку практики завідувач кафедри роз’яснює мету, завдання, зміст навчальної практики і вимоги до студентів під час її проходження. Керівник практики ознайомлює студентів з планом проходження практики, індивідуальними завданнями, вимогами до їх виконання і оформлення. До відома студентів доводиться також вид звітності про проходження практики (термін подання звіту та його форма).

Кожний студент отримує індивідуальні завдання на розробку програм для роботи з даними та організації віконного інтерфейсу користувача. Для розробки ускладнених програм студенти об’єднуються у групи спільної роботи.

Перед виконанням індивідуальних завдань студенти уже освоїли курс "Алгоритмізація і програмування", підчас вивчення якого здобували необхідні знання на лекційних і лабораторних заняттях, а також під час самостійної роботи над додатковими літературними джерелами. Керівник практики контролює процес розв’язання задач студентами та надає їм необхідні консультації.

Мета і зміст практики

Мета навчальної практики — поглибити знання, уміння та навички, здобуті при вивченні фундаментальних навчальних дисциплін "Алгоритмізація і програмування", "Вища математика", "Дискретна математика".

Завдання практики полягає у формуванні студентами початкових навичок професійної діяльності, пов’язаної з розробкою програм для розв’язання різних типів задач, підготовкою супровідної документації та поглибленням професійних знань і навичок, необхідних для практичної роботи прикладного програміста.

Організація і проведення практики

У комп’ютерних класах кафедри інформаційних технологій Уманського НУС студенти виконують завдання навчальної практики і подають звіти в зазначені терміни. Керівник навчальної практики зобов’язаний формулювати індивідуальні завдання, надавати студентам допомогу, необхідну для виконання самостійної роботи, перевіряти звіти, складати відгуки з практики.

Тривалість навчальної практики — один тиждень (30 академічних годин). З них 27 годин відводяться на практичну роботу студентів і 3 – на вирішення організаційних питань.

План-графік навчальної практики наведено нижче у таблиці.

План-графік навчальної практики

Тема	Години
1. Інструктаж з проведення практики та техніки безпеки	1
2. Створення алгоритму розв'язання задачі	6
5. Написання програми на мові C# для реалізації логіки задачі	6
3. Проектування інтерфейсу користувача і розробка екранних форм	6
7. Написання програми на мові C# для реалізації інтерфейсу	9
8. Остаточне оформлення щоденника та звіту про навчальну практику	1
9. Підведення підсумків і виставлення оцінок за практику	1
Всього:	**30**

Індивідуальні завдання включають розробку алгоритмів, програм (програмних кодів), опис програм і допоміжні матеріали та вказівки.

Програма навчальної практики

Передбачається, що у процесі проходження навчальної практики кожний студент має спроектувати і створити у середовищі Visual Studio завершений застосунок для розв'язування однієї з запропонованих задач.

Вся практична робота виконується у формі послідовного виконання окремих завдань, які у своїй сукупності відтворюють весь процес розробки програмного продукту. Основними етапами практики є: (1) вибір методу та розробка алгоритму розв'язування задачі, (2) написання програми на мові C# та (3) розробка і практична реалізація інтерфейсу за технологією Windows Forms.

Результати виконання завдань повинні бути оформлені у відповідності до розділу "Оформлення та оцінювання результатів практики" цього методичного посібника.

Завдання для проходження практики

Для виконання завдань практики створюються групи по 2-3 студенти, у які вони об'єднуються за узгодженням з керівником навчальної практики. На групу виді-

ляється один варіант завдання. Керівник практики призначає старшого групи, який організує і розподіляє роботи з проходження практики своєю групою.

При виконанні завдань кожний студент виконує призначену йому частину роботи самостійно. Частиною роботи може бути один з її завершених етапів: розробка і програмування класу (наприклад, розрахунок функції/методу або множини функцій), створення і програмування подій окремої екранної форми або її частини.

Старший групи слідкує за тим, щоб роботи виконувались у відповідності до завдання і таким чином, щоб їх результати могли бути зведені до єдиного застосунку і працювали узгоджено.

Завдання полягають у проектуванні та практичній реалізації автоматизованих робочих місць (АРМ) зі спрощеним функціоналом.

Варіанти завдань

1. АРМ менеджера з закупки та реалізації продукції у фермерів.

Вхідна інформація: Агенти з закупки передають інформацію про кількість продукції за видами, а магазини і заводи – про потреби.

Вихідна інформація: Звіт про рух продукції за видами продукції, Звіт про продаж продукції за видами та споживачами

Менеджер повинен знати, скільки і чого у нього є і формувати та відправляти партії потрібних обсягів споживачам.

2. АРМ менеджера бази з торгівлі нафтопродуктами.

Вхідна інформація: Обсяги надходження продукції у тонах; поточна температура; об'єм кожної автоцистерни кожного покупця; ціна 1 кг кожного нафтопродукту

Вихідна інформація: Звіт про рух продукції в тонах, Звіт про відпуск нафтопродуктів у тонах в розрізі покупців

Менеджер повинен слідкувати за наявністю нафтопродуктів у кожній ємності зберігання, вчасно робити замовлення, відпускати продукцію водіям автоцистерн з урахуванням їх обсягу і поточної температури.

3. АРМ працівника деканату.

Вхідна інформація: Відомості та екзаменаційні листи про здачу/перездачу екзаменів та заліків

Вихідна інформація: Звіти про успішність кожного студента, кожної групи в цілому, середні бали по викладачу, дисципліні, групі, курсу

Працівник повинен вести облік оцінок і періодично видавати звіти керівництву деканату

4. АРМ менеджера молочної ферми.

Вхідна інформація: Кількість дійних корів у групах, кількість доярок, кількість отриманого молока по групах за кожне доїння, жирність молока (у відсотках).

Вихідна інформація: Звіт про кількість і жирність молока в розрізі груп корів за кожне доїння; Звіт про отримання продукції у розрізі доярок; Звіт про трійку кращих доярок за кількістю отриманого молочного жиру

Менеджер повинен щоденно призначати доярок на групи корів (два доїння в день), обліковувати їх роботу і видавати звіти про вихід продукції

5. АРМ менеджера механізованої бригади.

Вхідна інформація: Прізвища трактористів; список планових робіт з коефіцієнтами переведення в гектари умовної оранки; список агрегатів; список полів

Вихідна інформація: Звіти про обсяги робіт (в га ум. оранки) по полях, видах робіт і трактористах

Менеджер повинен призначати трактористів на роботи, агрегати і поля та вести облік виконання робіт

6. АРМ менеджера цеху з виготовлення деталей.

Вхідна інформація: Планове завдання на цех, завдання кожному робітнику, обсяг виконаних ними робіт

Вихідна інформація: Звіти про виконання планів у розрізі деталей та робітників

Менеджер повинен щоденно розподіляти планові роботи між наявними робітниками, обліковувати виконані роботи і забезпечувати видачу звітів

7. АРМ менеджера зі збирання урожаю в саду.

Вхідна інформація: Список бригад збирачів і їх фактична чисельність на поточний день; список кварталів саду і залишкові обсяги роботи в них за окремими сортами плодів

Вихідна інформація: Звіти про обсяги зібраних плодів за їх видами, сортами, кварталами саду та бригадами; Список трьох кращих бригад за процентом виконання плану

Менеджер повинен розподіляти роботи між бригадами, встановлювати денну норму робіт і контролювати залишкові обсяги робіт про кварталах саду.

Оформлення та оцінювання результатів практики

Під час проходження практики студент веде щоденник, який подає разом зі звітом керівнику практики після її завершення.

Навчальна практика завершується оформленням індивідуальних письмових звітів про виконання програми практики та індивідуальних завдань.

У змістовній частині звіту описуються постановка задач, етапи дослідження та результати виконання запланованих завдань навчальної практики з детальним описом розроблених програм, тестовими прикладами та результатами їх програмної реалізації.

Тексти розроблених студентами програм та їх супровідна документована інформація подаються на дискеті разом зі звітом, оформленим на аркушах паперу формату А4, які нумерують, скріплюють і подають для підпису керівнику практики.

Керівник практики оцінює повноту виконання всіх завдань, глибину опанування методів розв'язання вибраних задач і якість створеного застосунку. Керівник оцінює також рівень самостійності роботи студента та його обізнаність з тими елементами застосунку, які ним не розроблялись.

Оцінка за практику вноситься в заліково-екзаменаційну відомість і в залікову книжку студента за підписом керівника практики.

Студент, який не виконав програму практики, може пройти її повторно в терміни, встановлені деканом (або завідувачем кафедри).

Якість виконання програми практики оцінюється за стобальною системою. Якщо щоденник або звіт про практику не подані, виставляється незадовільна оцінка.

Методичні рекомендації з виконання завдань навчальної практики

1. Створення графічного застосунку *Windows Forms*

Для створення графічних інтерфейсів за допомогою платформи .NET застосовуються різні технології - *Window Forms (WF)*, WPF, додатки для магазину *Windows Store* (для ОС Windows 8 / 8.1 / 10). Однак найбільш простий і зручною платформою досі залишається *Window Forms* або форми. Даний посібник має на меті дати розуміння принципів створення графічних інтерфейсів за допомогою технології *WinForms* і роботи основних елементів управління.

Створення застосунку WF у Visual Studio 2019 показане на рис. 1.

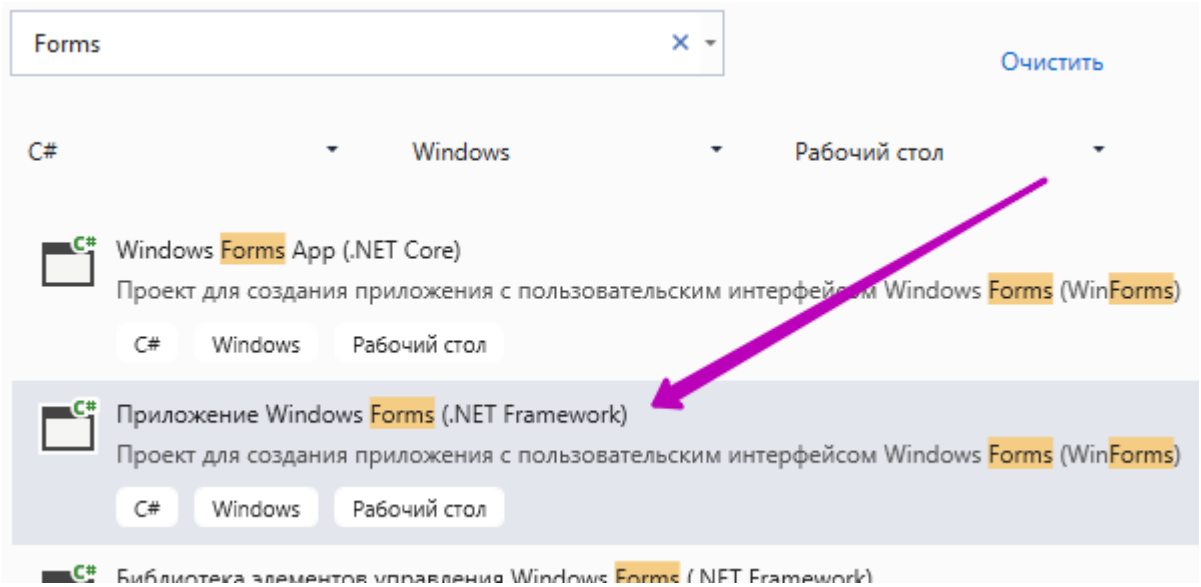

Рис. 1. Створення застосунку за технологією WF

Після створення проекту Visual Studio відкриє його зі створеними за замовчуванням файлами (рис. 2).

Більшу частину вікна проекту займає графічний редактор, який містить форму майбутнього застосунку. Спочатку ця форма пуста і має назву Form1. У

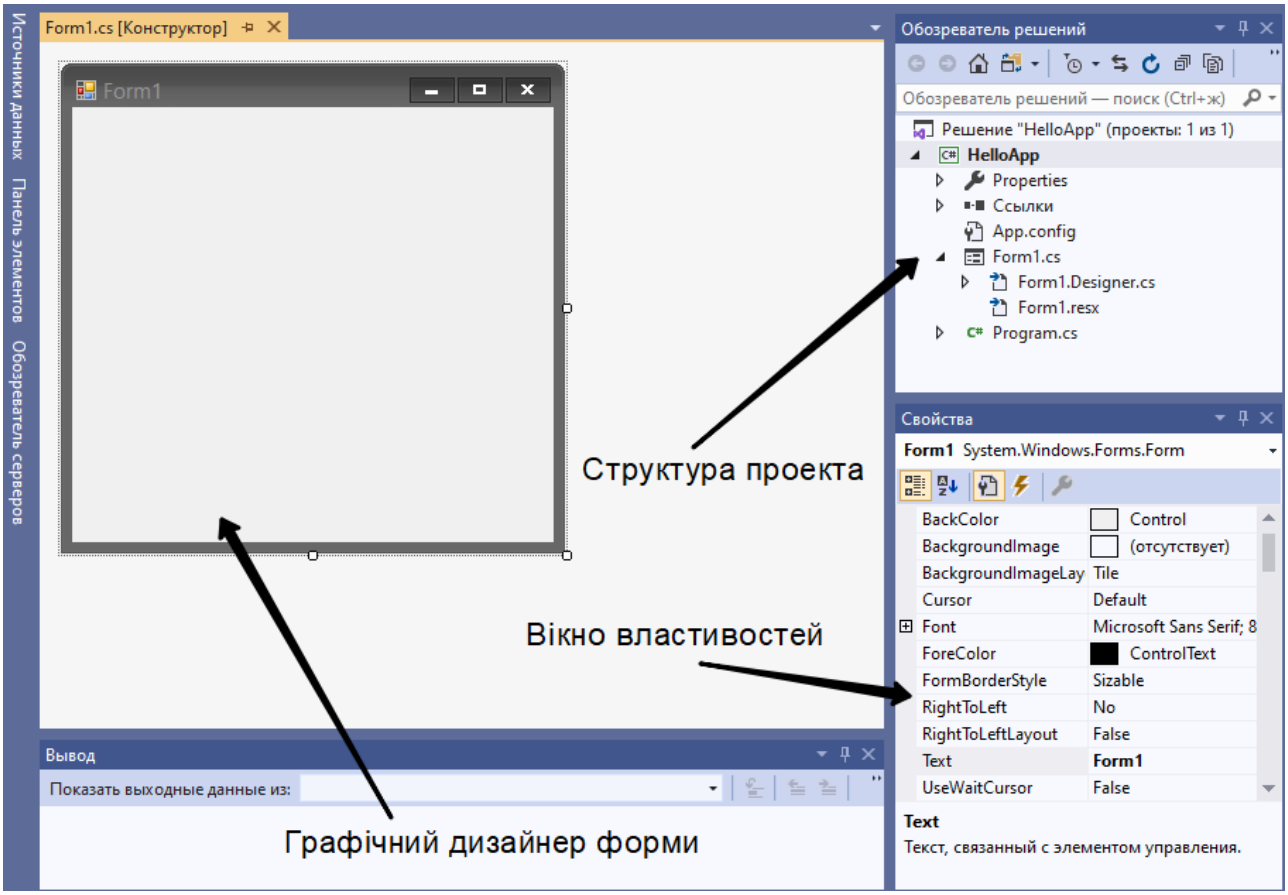

Рис. 2. Початкове вікно проекту, створеного за технологією WF

вікні огляду вмісту проекту міститься перелік всіх файлів, що складають проект. Крім уже відомих нам файлів, що входили до складу консольного застосунку, з'явились файли, що описують загальний вигляд і функціональність екранної форми, яка тепер також є невід'ємною частиною проекту.

Великого значення у налаштуванні роботи застосунку відіграє вікно Властивості (справа внизу), яке містить повний набір параметрів вікна форми або того її елемента, який виділений у даний час в графічному редакторі. Крім того, тут відображаються властивості і інших елементів проекту (крім форми). Наприклад, щоб змінити заголовок форми (зараз він "Form1"), потрібно виділити саму форму (на рис. 2 вона зараз виділена), знайти у вікні Властивості опцію Text і змінити її значення на потрібне (наприклад, "Hello, World!").

Функціональність екранної форми в основному пов'язана з застосуванням розміщених на її поверхні ***елементів управління*** (***Controls***).

Для розміщення елементів управління на формі служить спеціальна вкладка – ***Панель елементів***, або ***ToolBox***.Вона розташована зліва від графічного дизайнера (рис. 3).

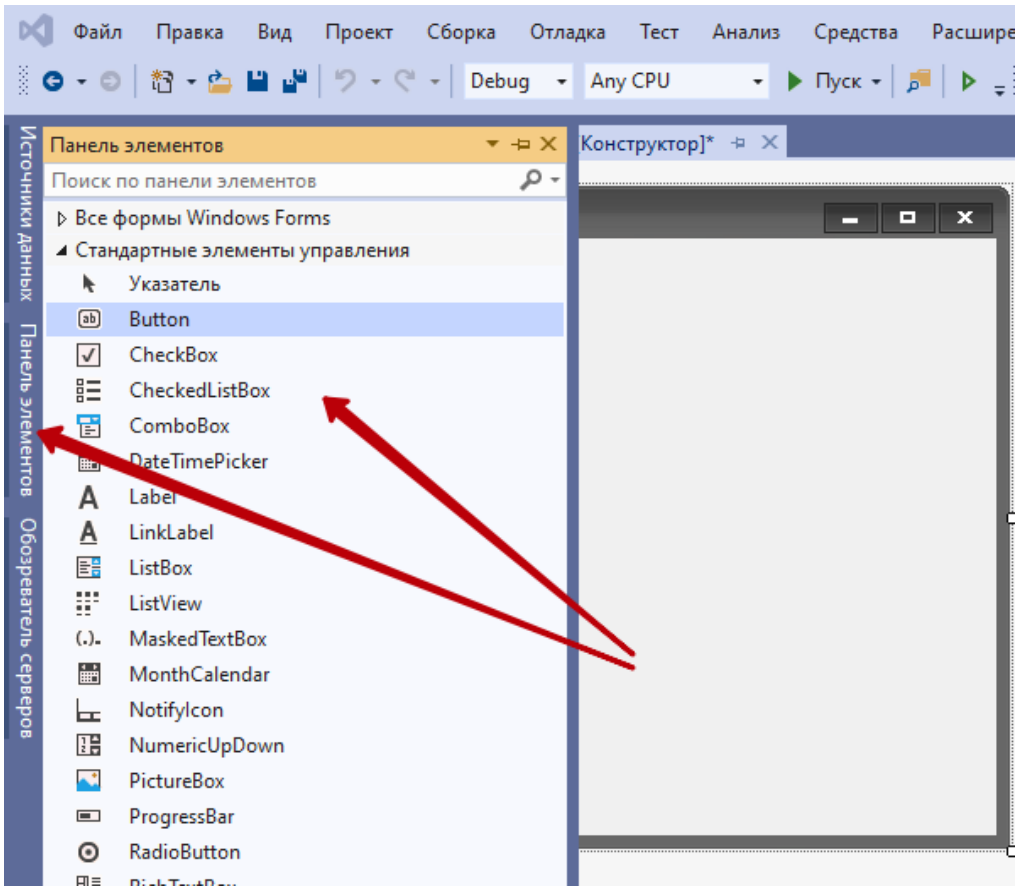

Рис. 3. Панель елементів управління форми

Щоб нанести елемент управління на форму, потрібно вибрати його на панелі елементів і клацнути мишкою по поверхні форми. Потім його можна перемістити та змінити його розмір.

Для наданні формі функціональності (тобто, щоб заставити її виконувати певну корисну роботу), з формою та її складовими має бути пов'язаний програмний код. Для переходу до нього можна натиснути на поверхні форми праву клавішу мишки і вибрати View Code. Є і інший спосіб: нанести на форму, наприклад, кнопку (Button) і двічі по ній клацнути мишкою. Відкриється вікно коду форми з програмами, що будуть виконуватись при відкритті форми (ініціалізація самої форми) і при клацанні по доданій кнопці (button1_Click). Додамо у програму для кнопки код, що виводить повідомлення (Лістинг 1).

Лістинг 1. Код для кнопки

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Привіт!");
        }
    }
}
```

Тепер у класі *Form1* міститься конструктор для створення форми і метод для реагування на просте натискання на кнопку *button1*.

2. Форма

Форма – основний будівельний блок застосунку. Через неї ми вводимо вихідні дані і отримуємо результати, а також виконуємо передбачені програмою операції, переважно використовуючи розташовані на формі елементи управління.

Взаємодія форми з користувачем здійснюється через механізм подій. Своїми діями користувач створює події, а окремі частини програми відзиваються на ці події, виконуючи певні набори операторів та інструкцій. Таким чином реалізується процес спілкування користувача і програми у процесі розв'язування задачі.

Система засобів для спілкування людини і комп'ютера у процесі розв'язування задачі називається ***інтерфейсом***.

Проект, що базується на використанні інтерфейсу у вигляді форм, має структуру, наведену на рис. 4. Хоч при запуску застосунку ми бачимо лише візуальні компоненти системи, насправді форма приховує потужний функціонал, який складається з методів, подій, властивостей тощо.

Після запуску застосунку (наприклад, F5) виконується метод *Main* (він розміщений у класі *Program*), який після кількох підготовчих процедур передає управління єдиній (поки що) створеній ним формі. Після виконання методу *Application.Run(new Form1)* форма стає активною (рис. 5). Якщо нам потрібно на старті застосунку запускати якусь іншу форму, то тут потрібно вказати назву її класу (звичайно, розробивши її попередньо).

Форма має досить складну структуру. У її складі має бути файл *Form1.Designer.cs*, який частиною класу *Form1* (про це свідчить рядок *partial class Form1*). Ця частина класу Form1 містить два методи: *Dispose*(), який виконує роль деструктора об'єкта, і *InitializeComponent*(), який встановлює початкові значення властивостей форми. Ці властивості можна змінювати у вікні *Властивості* вручну – тоді вони відобразяться у методі *InitializeComponent*, а можна вписувати в код метода вручну – тоді вони відобразяться у вікні *Властивості*. При додаванні елементів управління на форму вони також додаються у цей файл.

Файл Form1.resx зберігає ресурси форми. Як правило, ресурси використовуються для створення отнотипних форм відразу для декількох мовних культур.

Набагато більш важливий для нас файл Fjhv1.cs містить код або програмну логіку форми. Спочатку в ньому міститься тільки конструктор форми, який просто викликає метод *InitializeComponent*(), однак, по мірі створення подій їх обробники будуть з'являтись саме тут (див. наприклад, Лістинг 1, у якому

додана подія Click для створеної для прикладу кнопки button1). В подальшому нас буде цікавити переважно саме цей файл.

3. Основні властивості форм

За допомогою спеціального вікна *Properties* (Властивості), розташованого праворуч від дизайнера форм, *Visual Studio* надає нам зручний інтерфейс для управління властивостями елемента.

Основні властивості форми:

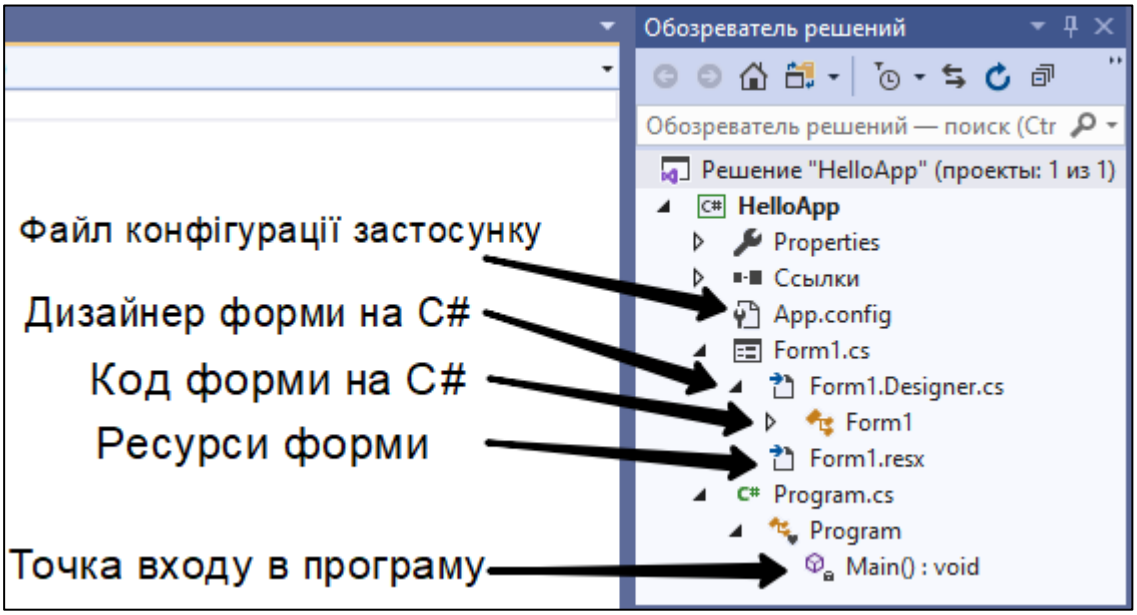

Рис. 4. Компоненти застосунку WF

```
1   using System;
2   using System.Collections.Generic;
3   using System.Diagnostics;
4   using System.Linq;
5   using System.Threading;
6   using System.Windows.Forms;
7
8   namespace HelloApp
9   {
10      static class Program
11      {
12          /// <summary> Главная точка входа для приложения.
15          [STAThread]
16          static void Main()
17          {
18              Application.EnableVisualStyles();
19              Application.SetCompatibleTextRenderingDefault(false);
20              Application.Run(new Form1());
21          }
22      }
23  }
```

Рис. 5. Точка входження застосунку – метод *Main*

- *Name*: встановлює ім'я форми – точніше ім'я класу, який успадковується від класу *Form*
- *BackColor*: вказує на фоновий колір форми. Клацнувши на це властивість, ми зможемо вибрати той колір, який нам підходить зі списку запропонованих кольорів або кольорової палітри
- *BackgroundImage*: вказує на фонове зображення форми
- *BackgroundImageLayout*: визначає, як зображення, задане у властивості *BackgroundImage*, буде розташовуватися на формі.

- *ControlBox*: вказує, чи відображається меню форми. В даному випадку під меню розуміється меню самого верхнього рівня, де знаходяться іконка програми, заголовок форми, а також кнопки мінімізації форми і хрестик. Якщо ця властивість має значення *false*, то ми не побачимо ні іконку, ні хрестика, за допомогою якого зазвичай закривається форма
- *Cursor*: визначає тип курсора, який використовується на формі
- *Enabled*: якщо ця властивість має значення *false*, то форма не зможе отримувати будь-які введення від користувача, тобто ми не зможемо натиснути на кнопки, ввести текст в текстові поля тощо
- *Font*: задає шрифт для всієї форми і всіх розміщених на ній елементів управління. Однак, задавши у елементів форми свій шрифт, ми можемо тим самим перевизначити його
- *ForeColor*: колір шрифту на формі
- *FormBorderStyle*: вказує, як буде відображатися межа форми і рядок заголовка. Встановлюючи ця властивість в *None* можна створювати зовнішній вигляд програми довільної форми
- *HelpButton*: вказує, чи відображається кнопка довідки форми
- *Icon*: задає іконку форми
- *Location*: визначає положення по відношенню до верхнього лівого кута екрана, якщо для властивості *StartPosition* встановлено значення *Manual*
- *MaximizeBox*: вказує, чи буде доступна кнопка максимізації вікна в заголовку форми
- *MinimizeBox*: вказує, чи буде доступна кнопка мінімізації вікна
- *MaximumSize*: задає максимальний розмір форми
- *MinimumSize*: задає мінімальний розмір форми
- *Opacity*: задає прозорість форми
- *Size*: визначає початковий розмір форми
- *StartPosition*: вказує на початкову позицію, з якою форма з'являється на екрані
- *Text*: визначає заголовок форми
- *TopMost*: якщо ця властивість має значення *true*, то форма завжди буде знаходитися поверх інших вікон
- *Visible*: видима чи форма, якщо ми хочемо приховати форму від користувача, то можемо поставити даній властивості значення *false*
- *WindowState*: вказує, в якому стані форма буде знаходитися при запуску: в нормальному, максимізувати або мінімізованому

Програмна настройка властивостей

За допомогою значень властивостей у вікні *Властивості* ми можемо змінити на свій розсуд зовнішній вигляд форми, але все те ж саме ми можемо зробити динамічно в коді. Перейдемо до коду, для цього натиснемо правою кнопкою миші на формі і виберемо в контекстному меню *View Code* (Перегляд коду). Перед нами відкривається файл коду *Form1.cs*. Змінимо його, як показано на рис. 6 (при умові, що створена нами раніше вручну кнопка *button1* не видалялась і залишилась на формі).

Ключове слово this використовується для звертання до створюваного конструктором екземпляру форми.

Установка розмірів форми

Для установки розмірів форми можна використовувати такі властивості як *Width*/*Height* або *Size*. *Width*/*Height* приймають числові значення, як в наведе-

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace HelloApp
12  {
13      public partial class Form1 : Form
14      {
15          public Form1()
16          {
17              InitializeComponent();
18              Text = "Hello World!";
19              this.BackColor = Color.Aquamarine;
20              this.Width = 250;
21              this.Height = 250;
22              this.button1.Top= 150;
23              this.button1.Left = 120;
24          }
25
26          private void button1_Click(object sender, EventArgs e)
27          {
28              BackColor = Color.Azure;
29              MessageBox.Show("Привіт!");
30          }
31      }
32  }
```

Рис. 6. Програма у файлі *Form1.cs* після внесення змін

ному вище прикладі. При установці розмірів через властивість *Size*, нам треба привласнити властивості об'єкт типу *Size*:

```
this.Size = new Size(200,150);
```

Початкове розташування форми

Початкове розташування форми встановлюється за допомогою властивості *StartPosition*, яке може приймати одне з таких значень:

- *Manual*: Положення форми визначається властивістю *Location*
- *CenterScreen*: Положення форми в центрі екрана
- *WindowsDefaultLocation*: Позиція форми на екрані задається системою *Windows*, а розмір визначається властивістю *Size*
- *WindowsDefaultBounds*: Початкова позиція і розмір форми на екрані задається системою *Windows*
- *CenterParent*: Положення форми встановлюється в центрі батьківського вікна

Всі ці значення містяться в переліку *FormStartPosition*, тому, щоб, наприклад, встановити форму в центрі екрану, нам треба прописати так:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

Фон і колір форми

Щоб встановити колір як фону форми, так і шрифту, нам треба використовувати колірне значення, що зберігається в структурі *Color*:

```
this.BackColor = Color.Aquamarine;
this.ForeColor = Color.Red;
```

Крім того, ми можемо в якості фону задати зображення у властивості *BackgroundImage*, вибравши його у вікні властивостей або в коді, вказавши шлях до зображення:

```
this.BackgroundImage =
          Image.FromFile("C:\\Users\\Students\\Pictures\\3332.jpg");
```

Щоб належним чином налаштувати потрібний нам відображення фонової картинки, треба використовувати властивість *BackgroundImageLayout*, яке може приймати одне з наступних значень:

- *None*: Зображення поміщається в верхньому лівому куті форми і зберігає свої початкові значення
- *Tile*: Зображення розташовується на формі у вигляді мозаїки
- *Center*: Зображення розташовується по центру форми
- *Stretch*: Зображення розтягується до розмірів форми без збереження пропорцій

- *Zoom*: Зображення розтягується до розмірів форми зі збереженням пропорцій

Наприклад, розташуємо форму в центрі екрану:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

4. Додавання форм та взаємодія між формами

Щоб додати ще одну форму в проект, натиснемо на ім'я проекту у вікні *Solution Explorer* (Оглядач рішень) правою кнопкою миші і виберемо *Add* (Додати) -> *Windows Form*.

Під час створення нової форми вкажемо її назву, наприклад, *Form2*. Тепер у структурі нашого проекту з'явився новий клас (рис. 7).

Для виклику другої форми з першої змінимо код для створеної нами раніше кнопки button1 (Лістинг 2).

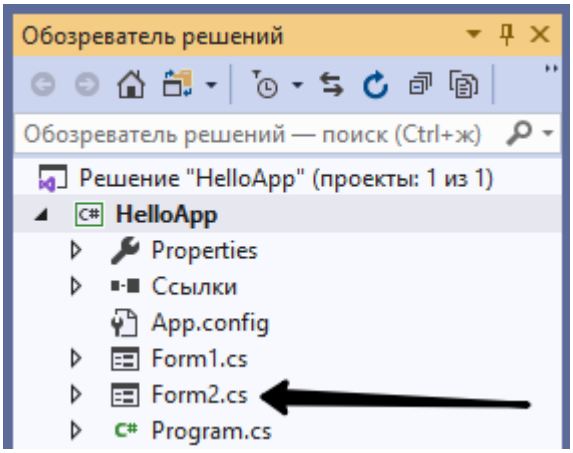

Рис. 7. Новий клас (*Form2*)

Лістинг 2. Новий код для кнопки

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 newForm = new Form2();
    newForm.Show();
}
```

Код створює об'єктну змінну *newform* і викликає для неї метод *Show*(). Ми цього методу не створювали; його взято зі стандартного класу *Form*, від якого успадкований наші класи форм (і *Form1*, і *Form2*). Сам оператор успадкування має вигляд двокрапки, наприклад:

```
public partial class Form1 : Form
{
    ...
}
```

Тепер спробуємо добитись, щоб друга форма впливала на першу. Для цього зробимо дві речі: (а) будемо передавати об'єкт першої форми у конструктор другої і (б) створимо у класі другої форми конструктор, який приймає такий об'єкт і маніпулює ним.

Перше виявляється дуже простим – ключове слово *this* якраз і представляє посилання на об'єкт, на якому розташована кнопка, для якої ми пишемо код. Його нам і потрібно передати другій формі:

```
private void button1_Click(object sender,
          EventArgs e)
{
    Form2 newForm = new Form2(this);
    newForm.Show();
}
```

Друге завдання потребує створення додаткового конструктора в класі *Form2*:

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
    }
    public Form2(Form1 f)
    {
        InitializeComponent();
        this.BackColor = Color.Yellow;
    }
}
```

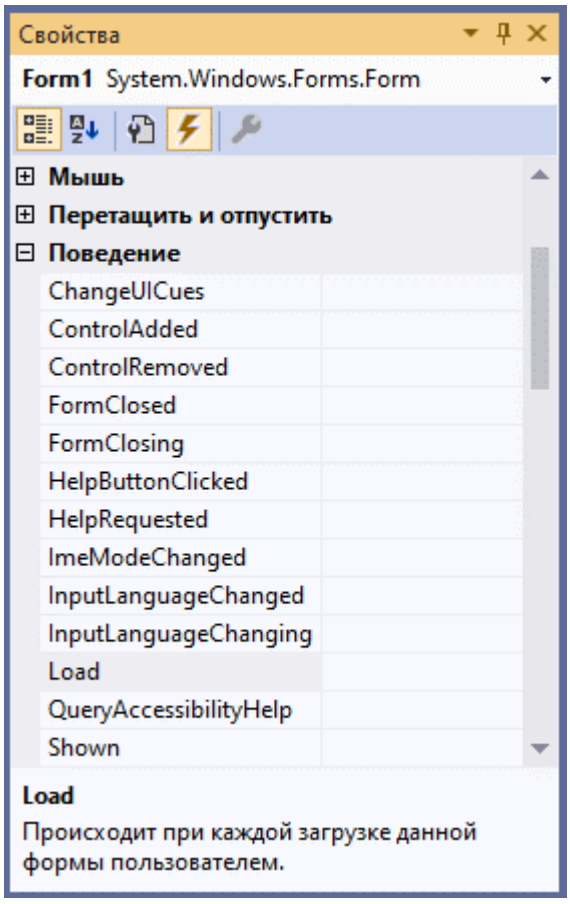

Рис. 8. Фрагмент списку подій форми

Цей новий конструктор (*public Form2*(*Form1 f*){}) отримує переданий вище об'єкт першої форми і змінює йому колір фону на жовтий.

При роботі з декількома формами треба враховувати, що одна з них є головною – яка запускається першою в файлі *Program.cs*. Якщо у нас одночасно відкрито декілька форм, то при закритті головної закривається весь додаток і разом з ним всі інші форми.

5. Події у *Windows Forms*

Для взаємодії з користувачем у *Windows Forms* використовується механізм подій. Події в *Windows Forms* представляють стандартні події C#, з тією відмінністю, що застосовуються до візуальних компонентів і підкоряються тим же правилам, що події в C#. Але створення обробників подій в *Windows Forms* все ж має деякі особливості.

Перш за все в *WinForms* є певний стандартний набір подій, який здебільшого є у всіх візуальних компонентів. Окремі елементи додають свої події, але принципи роботи з ними будуть схожі. Щоб подивитися всі події елемента, пот-

рібно вибрати цей елемент в полі графічного дизайнера і перейти до вкладки подій на панелі форм. Наприклад, події форми наведені на рис. 8. Тут виділена подія *Load* (*Завантаження*), тому внизу вікна властивостей виведене пояснення саме для неї.

Для виведення списку подій у заголовковій частині вікна Властивості необхідно натиснути кнопку зі значком у формі блискавки.

Щоб додати обробник події, тобто програмний код, що буде автоматично запускатись і виконуватись щоразу, коли настає відповідна подія, потрібно двічі клацнути по пустому полю у вікні Властивості справа від поля з назвою події (у нас зараз це Load). Система створить у файлі *Form1.cs* обробника події з назвою, що є комбінацією назви об'єкта, для якого він створюється і (через знак підкреслення) назви самої події. У нашому випадку це *Form1_Load*. Крім того, здійсниться автоматичний перехід до створеної підпрограми для заповнення її потрібним текстом коду:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {

    }
}
```

Після написання цього програмного, коду він буде виконуватись кожного разу, коли форма буде завантажуватись.

Як правило, обробники подій різних візуальних компонентів мають два параметри:

- ✓ *object sender* – об'єкт, що ініціював подію;
- ✓ *EventArgs e* – аргумент, що містить інформацію про подію.

Так відбувається створення самого обробника події. Однак, крім цього, у файлі Form1.Designer.cs здійснюється ще одна дія, а саме додавання створеного обробника до набору подій об'єкта:

```
private void InitializeComponent()
{
   ...
   this.Load += new System.EventHandler(this.Form1_Load);
   ...
}
```

Тому, якщо потрібно буде видалити створений обробник події *Load*, то мало просто знищити його код у *Form1.cs*, а ще й потрібно видалити рядок з прив'язкою цього обробника з файлу *Form1.Designer.cs*. Інакше система буде шукати видалений фрагмент і, не знайшовши, викине відповідний виняток.

Такий ускладнений механізм має одну величезну перевагу – він дозволяє додавати кілька обробників подій і додавати їх ***програмно***. Наприклад, ми можемо додавати свої власні обробники з довільними іменами у різні частини програми. Для прикладу додамо обробник події у конструктор форми *Form1*:

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();
      this.Load += LoadEvent;
   }
   private void Form1_Load(object sender, EventArgs e)
   {
      this.Text = ":: Перша форма ::";
   }
   private void LoadEvent(object sender, EventArgs e)
   {
      this.BackColor = Color.Aquamarine;
      this.Width = 450;
      this.Height = 250;
      this.button1.Top = 150;
      this.button1.Left = 320;
   }
   ...
}
```

Тепер у події *Load* два обробники. Один з них додається у розглянутому нами раніше файлі *Form1.Designer.cs*, а другий – у конструкторі форми.

Для закриття форми потрібно створити кнопку, передбачивши для неї подію Click з єдиною інструкцією:

```
this.Close();
```

При натисканні на цю кнопку закриється і ця форма, і Form2, якщо на цей момент вона буде відкрита.

Для переміщення форми обробляються дві події форми – подія натискання кнопки миші і подія переміщення покажчика миші.

6. Динамічне додавання елементів форми

Для організації елементів управління в пов'язані групи існують спеціальні елементи – контейнери. Прикладами таких контейнерів можуть бути *Panel*, *FlowLayoutPanel*, *SplitContainer*, *GroupBox*. Ту ж форму також можна віднести до контейнерів. Використання контейнерів полегшує управління елементами, а також надає формі певного візуального стилю.

Всі контейнери мають властивість *Controls*, яка містить всі елементи даного контейнера. Коли ми переміщуємо якийсь елемент з панелі інструментів на контейнер, наприклад, кнопку, вона автоматично додається в дану колекцію даного контейнера. Ми також можемо додати елемент керування в цю ж колекцію динамічно за допомогою коду.

Нехай нам потрібно додати кнопку на поверхню форми. Ми уже знаємо, що це можна зробити в дизайнері форми вручну, перетягши кнопку з панелі елементів на форму і розташувавши її в потрібному місці. Потім у вікні Властивості можна надати їй потрібного виду і функціональності. Однак такий спосіб досить обмежений, оскільки часто потрібно динамічно створювати і видаляти, показувати і приховувати елементи на формі.

Для динамічного додавання елементів створимо обробник події завантаження форми в файлі коду і додамо до нього текст програми (Лістинг 3).

Лістинг 3. Динамічне додавання кнопки.

```
private void Form1_Load(object sender, EventArgs e)
{
    Button helloButton = new Button();
    helloButton.BackColor = Color.LightGray;
    helloButton.ForeColor = Color.DarkGray;
    helloButton.Location = new Point(10, 10);
    helloButton.Text = "Привет";
    this.Controls.Add(helloButton);
}
```

Спочатку ми створюємо кнопку і встановлюємо її властивості. Потім, використовуючи метод *Controls.Add* ми додаємо її в колекцію елементів форми. Якби ми це не зробили, ми б кнопку не побачили, оскільки в цьому випадку для нашої форми її просто не існувало б.

За допомогою методу *Controls.Remove* () можна видалити раніше доданий елемент з форми:

```
this.Controls.Remove(helloButton);
```

Хоча в даному випадку в якості контейнера використовувалася форма, але при додаванні і видаленні елементів з будь-якого іншого контейнера, наприклад, *GroupBox*, будуть застосовуватися ті ж самі методи.

7. Елементи GroupBox, Panel і FlowLayoutPanel

GroupBox являє собою спеціальний контейнер, обмежений від решти форми границею у вигляді лінії. Він має заголовок, який встановлюється через властивість *Text*. Щоб зробити *GroupBox* без заголовка, просто значенням властивості *Text* встановлюється порожній рядок.

Часто цей елемент використовується для групування перемикачів – елементів *RadioButton*, оскільки дозволяє розмежувати групи таких перемикачів.

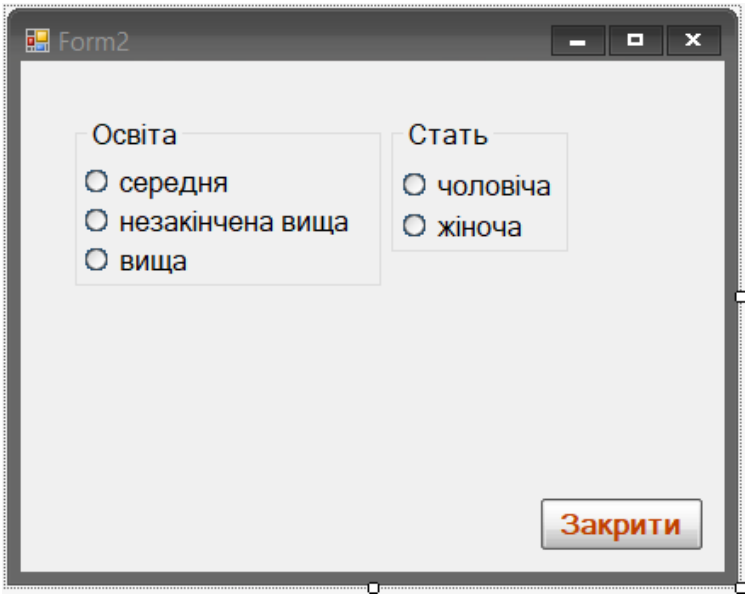

Рис. 9. Два контейнери *GroupBox* з п'ятьма елементами управління *RadioButton*

Panel. Елемент управління *Panel* являє собою панель і так само як і *GroupBox*, об'єднує елементи в групи. *Panel* може візуально зливатися з рештою форми, якщо має таке ж значення кольору фону у властивості *BackColor*, що і форма. Щоб її виділити можна, крім іншого кольору фону, вказати значення стилю границі, наприклад, Fixed3D (за допомогою властивості *BorderStyle*). За замовчуванням стиль має значення *None*, тобто відсутність границі.

Також якщо панель має багато елементів, які виходять за її межі, можна увімкнути прокручування панелі, встановивши її властивість *AutoScroll* в *true*.

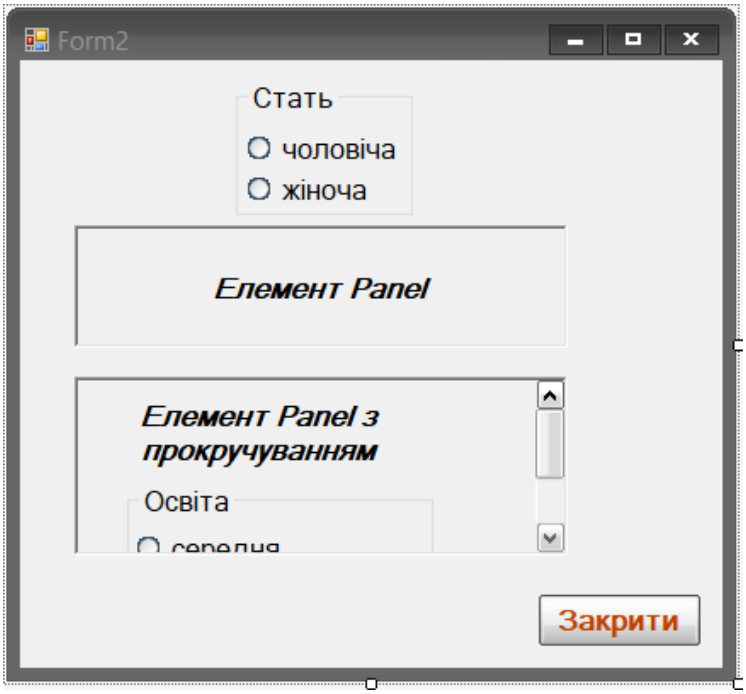

Рис. 10. Дві панелі – з прокруткою і без

Так само, як і форма, *GroupBox* і *Panel* мають "свої" колекції елементів, і можна динамічно додавати в ці контейнери нові елементи. Наприклад, на формі є елемент *GroupBox*, який має ім'я groupBox1:

```
private void Form1_Load(object sender, EventArgs e)
{
    Button helloButton = new Button();
    helloButton.BackColor = Color.LightGray;
    helloButton.ForeColor = Color.Red;
    helloButton.Location = new Point(30, 30);
    helloButton.Text = "Привет";
    groupBox1.Controls.Add(helloButton);
}
```

Для вказівки розташування елемента в контейнері використовується структура *Point*: *new Point (30, 30);*, через яку в конструктор передається розміщення по осях X і Y. Ці координати встановлюються щодо лівого верхнього кута контейнера – тобто в даному випадку елемента *GroupBox*.

При цьому треба враховувати, що контейнером верхнього рівня є форма, а елемент *groupBox1* сам міститься у колекції елементів форми. І при бажанні ми могли б видалити його:

```
this.Controls.Remove(groupBox1);
```

FlowLayoutPanel. Елемент *FlowLayoutPanel* є успадкованим від класу *Panel*, і тому успадковує всі його властивості. Однак при цьому до нього додана додаткова функціональність. Так, цей елемент дозволяє змінювати позиціонування і компоновку дочірніх елементів при зміні розмірів форми під час виконання програми.

Властивість елемента *FlowDirection* дозволяє задати напрямок, в якому спрямовані дочірні елементи. За замовчуванням має значення *LeftToRight* – тобто елементи будуть розташовуватися починаючи від лівого верхнього краю (рис. 11). Наступні елементи будуть йти вправо. Це властивість може набувати таких значень:

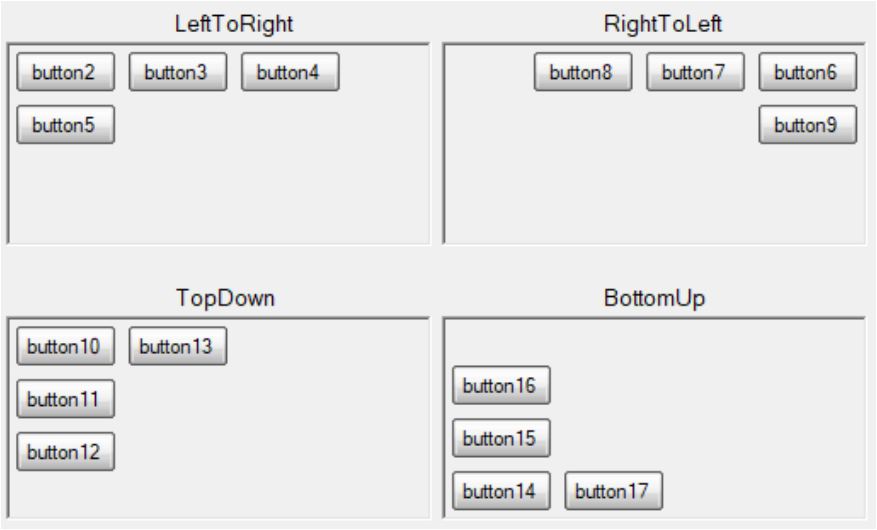

Рис. 11. Результат дії різних значень властивості *FlowDirection*

- *LeftToRight* – елементи розташовуються від лівого верхнього кута в правий бік;
- *RightToLeft* – елементи розташовуються від правого верхнього кута в лівий бік;
- *TopDown* – елементи розташовуються від лівого верхнього кута і йдуть вниз;
- *BottomUp* – елементи розташовуються від лівого нижнього кута і йдуть вгору.

При розташуванні елементів важливу роль відіграє властивість *WrapContents*. За замовчуванням воно має значення *True*. Це дозволяє переносити елементи, які не поміщаються в *FlowLayoutPanel*, на новий рядок або в новий стовпець. Якщо воно має значення *False*, то елементи не переносяться, а до контейнера просто додаються смуги прокрутки, якщо властивість *AutoScroll* має значення *True*.

8. Елемент TableLayoutPanel

Елемент *TableLayoutPanel* також перевизначає панель і має в своєму розпорядженні дочірні елементи управління у вигляді таблиці, де для кожного елемента є своя комірка. Якщо потрібно помістити в клітинку більш одного еле-

мента, то в цю комірку додається інший компонент *TableLayoutPanel*, в який потім вкладаються інші елементи.

Щоб встановити необхідну кількість рядки стовпців таблиці, ми можемо використовувати властивості *Rows* і *Columns* відповідно. Вибравши один з цих пунктів у вікні *Properties* (Властивості), нам відобразиться вікно для настройки стовпців і рядків (рис. 12).

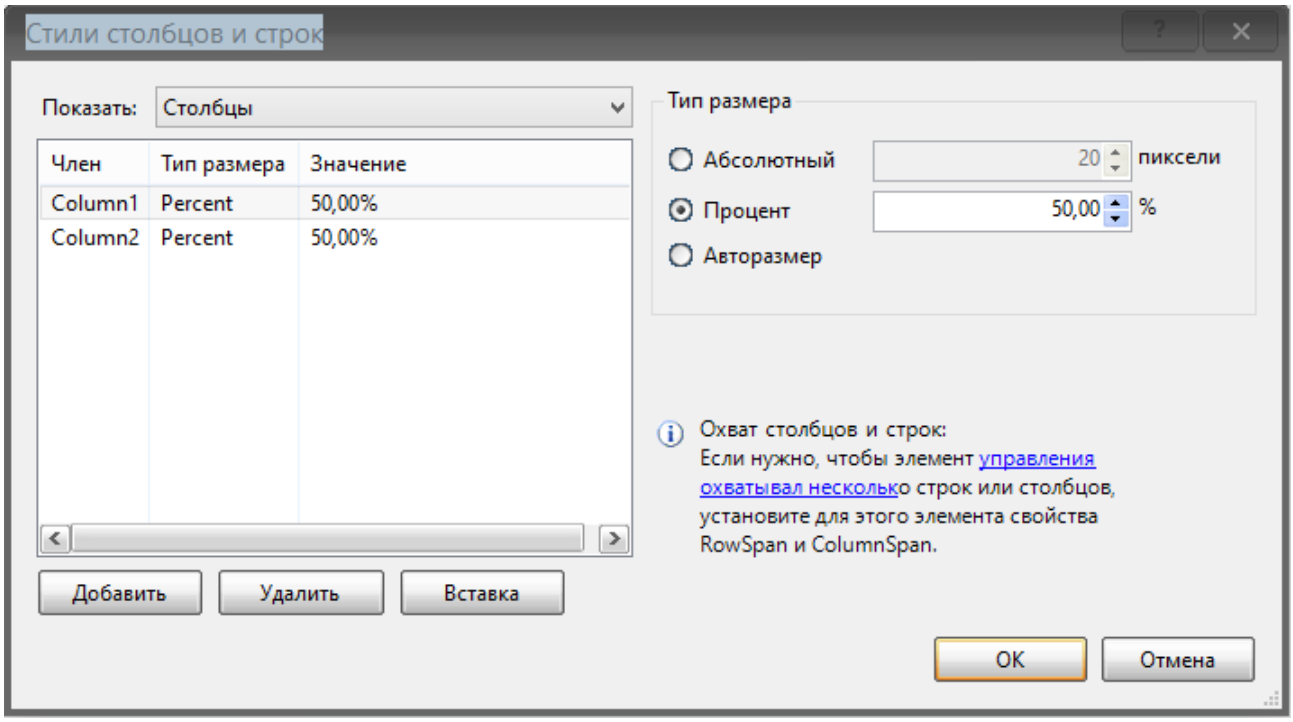

Рис. 12. Встановлення параметрів рядків і стовпців *TableLayoutPanel*

В поле *Size Type* можна вказати розмір стовпців/рядків. Доступні три можливі варіанти:

Absolute: задається абсолютна величина для рядків або стовпців в пікселях

Percent: задається відносний розмір у відсотках. Якщо потрібно створити гумовий дизайн форми, щоб її рядки і стовпці, а також елементи управління в осередках таблиці автоматично масштабувати при зміні розмірів форми, то потрібно використовувати саме цю опцію

AutoSize: висота рядків і ширина стовпців задається автоматично в залежності від розміру найбільшої в рядку або стовпці клітинки.

Також можна комбінувати ці значення, наприклад, один стовпець може бути фіксованим з абсолютною шириною, а інші – мати ширину у відсотках.

У цьому діалоговому вікні ми також можемо додати або видалити рядки і стовпці. У той же час, графічний дизайнер в *Visual Studio* не завжди відразу відображає зміни в таблиці (додавання або видалення рядків і стовпців, зміна їх розмірів). Тому, якщо змін на формі ніяких не відбувається, треба її закрити і потім відкрити в графічному дизайнері заново.

Отже, наприклад, у мене є три стовпці і три рядки розмір у яких однаковий і становить по 33.33%. У кожну клітинку таблиці додано кнопку, у якій для властивості *Dock* встановлено значення *Fill*. Якщо також і для елемента *TableLayoutPanel* для цієї властивості встановити значення *Fill*, то при зміні розмірів форми буде розтягуватись/стискатись таблиця і всі її кнопки (рис. 13). Це дуже зручно для створення масштабованих інтерфейсів.

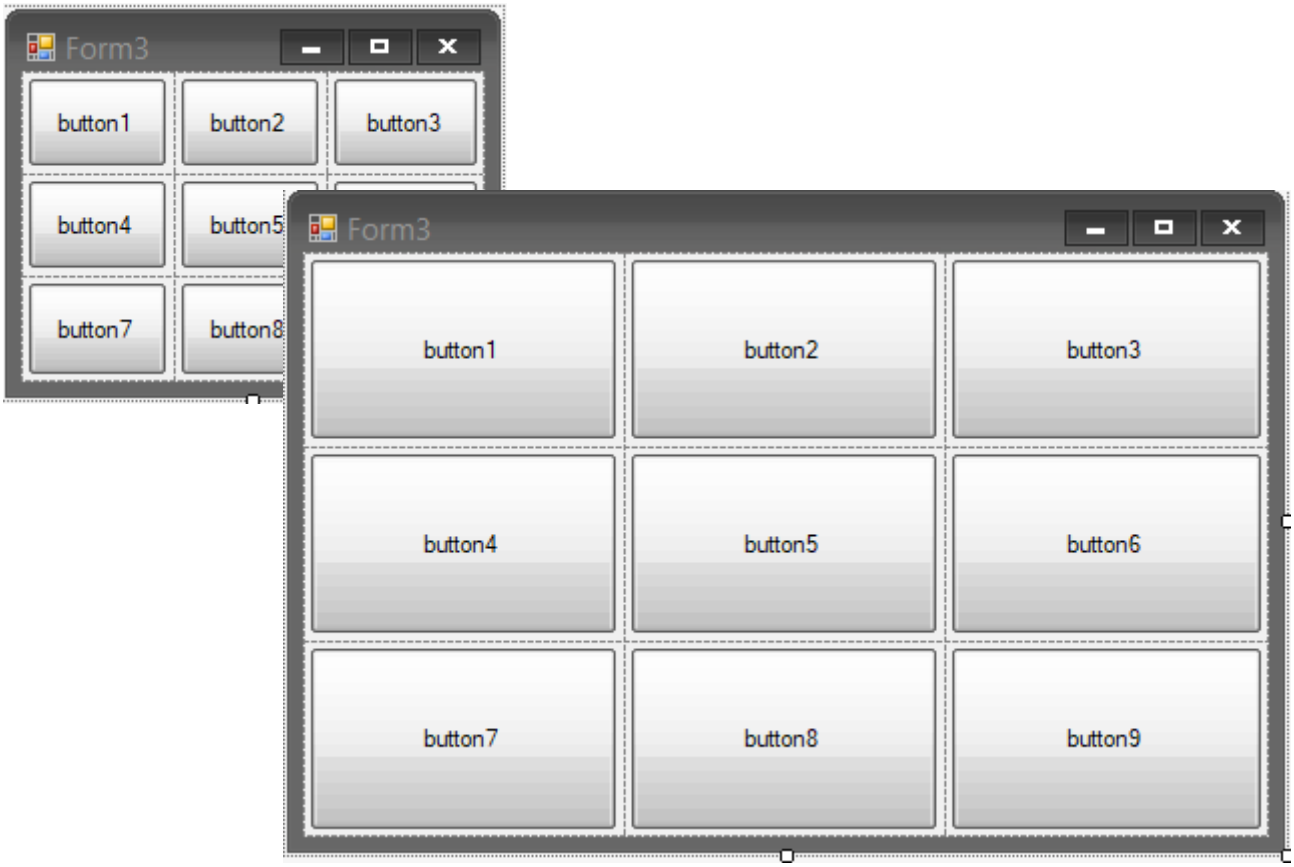

Рис. 13. Дія параметра *Dock* зі значенням *Fill* в елементі управління *TableLayoutPanel*

У коді ми можемо динамічно змінювати значення стовпців і рядків. Причому всі стовпці представлені типом *ColumnStyle*, а рядки – типом *RowStyle*:

```
tableLayoutPanel1.RowStyles[0].SizeType = SizeType.Percent;
tableLayoutPanel1.RowStyles[0].Height = 40;

tableLayoutPanel1.ColumnStyles[0].SizeType = SizeType.Absolute;
tableLayoutPanel1.ColumnStyles[0].Width = 50;
```

Щоб установити розмір в *ColumnStyle* і *RowStyle* визначено властивість *SizeType*, яке приймає одне із значень однойменного переліку *SizeType*.

Додавання елемента в контейнер *TableLayoutPanel* має свої особливості. Можна додати його в наступну вільну комірку або можна явно вказати клітинку таблиці:

```
Button saveButton = new Button();
// додаємо кнопку в наступну вільну комірку
tableLayoutPanel1.Controls.Add(saveButton);
// додаємо кнопку в комірку (2,2)
tableLayoutPanel1.Controls.Add(saveButton, 0, 2);
```

В даному випадку додаємо кнопку в клітинку, утворену на перетині першого стовпця і третього рядка. Правда, якщо у нас немає стільки рядків і стовпців, то система автоматично вибере потрібну клітинку для додавання.

9. Розміри елементів і їх позиціонування в контейнері

Позиціонування

Для кожного елемента управління ми можемо визначити властивість *Location*, яке задає координати верхнього лівого кута елемента відносно контейнера. При перенесенні елемента з панелі інструментів на форму ця властивість встановлюється автоматично. Однак потім у вікні *Властивості* можна вручну поправити координати положення елемента. Це можна зробити і програмно:

```
private void Form1_Load(object sender, EventArgs e)
{
    button1.Location = new Point(50, 50);
}
```

Установка розмірів

Розміри елемента можна задати за допомогою властивості *Size* у вікні *Властивості*. Додаткові властивості *MaximumSize* і *MinimumSize* дозволяють обмежити мінімальний і максимальний розміри.

Установка властивостей в коді:

```
button1.Size = new Size { Width = 50, Height = 25 };
// установка властивостей окремо
button1.Width = 100;
button1.Height = 35;
```

Властивість Anchor

Додаткові можливості по позиціонуванні елемента дозволяє визначити властивість *Anchor*. Це властивість визначає відстань між однією зі сторін елемента і стороною контейнера. І якщо при роботі з контейнером ми будемо його розтягувати, то разом з ним буде розтягуватися і вкладений елемент.

Автоматично у кожного елемента, що додається, ця властивість встановлюється рівною *Top*, *Left*. Це означає, що якщо ми будемо розтягувати форму вліво або вгору, то елемент збереже відстань від лівої і верхньої межі елемента до кордонів контейнера, в якості якого виступає форма.

Ми можемо поставити чотири можливих значення для цієї властивості або їх комбінацію:

- *Top*
- *Bottom*
- *Left*
- *Right*

Наприклад, якщо ми змінимо значення цієї властивості на протилежне - *Bottom*, *Right*, тоді у нас буде незмінною відстань між правою і нижньою стороною елемента і формою.

При цьому треба відзначити, що дана властивість враховує відстань до кордонів контейнера, а не форми. Тобто якщо у нас на формі є елемент *Panel*, а на *Panel* розташована кнопка, то на кнопку впливатиме зміна кордонів *Panel*, а не форми. Розтягування форми буде в цьому випадку впливати тільки, якщо воно впливає на контейнер *Panel*.

Щоб задати цю властивість в коді, треба використовувати перелік *AnchorStyles*:

```
button1.Anchor = AnchorStyles.Left;
// задаємо комбінацію значень
button1.Anchor = AnchorStyles.Left | AnchorStyles.Top;
```

Властивість Dock

Властивість *Dock* дозволяє прикріпити елемент до певної стороні контейнера. За замовчуванням воно має значення *None*, але також дозволяє задати ще п'ять значень:

- *Top*: елемент притискається до верхньої межі контейнера
- *Bottom*: елемент притискається до нижньої межі контейнера
- *Left*: елемент притискається до лівого боку контейнера
- *Right*: елемент прикріплюється до правого боку контейнера
- *Fill*: елемент заповнює весь простір контейнера

10. Панель вкладок TabControl и SplitContainer

TabControl

Елемент *TabControl* дозволяє створити елемент управління з декількома вкладками. Кожна вкладка буде зберігати деякий набір інших елементів управління, такі як кнопки, текстові поля тощо. Кожна вкладка представлена класом *TabPage*.

Щоб налаштувати вкладки елемента *TabControl* використовується властивість *TabPages*. При перенесенні елемента *TabControl* з панелі інструментів на форму за замовчуванням створюються дві вкладки - *tabPage1* і *tabPage2*. Змінюється їх відображення за допомогою властивості *TabPages*. Якщо вибрати її у вікні властивостей, відкриється спеціальне діалогове вікно (рис. 14).

Кожна вкладка представляє свого роду панель, на яку можна додавати інші елементи управління, а також заголовок, за допомогою якого з'являється можливість перемикатися по вкладках. Текст заголовка задається за допомогою властивості *Text*.

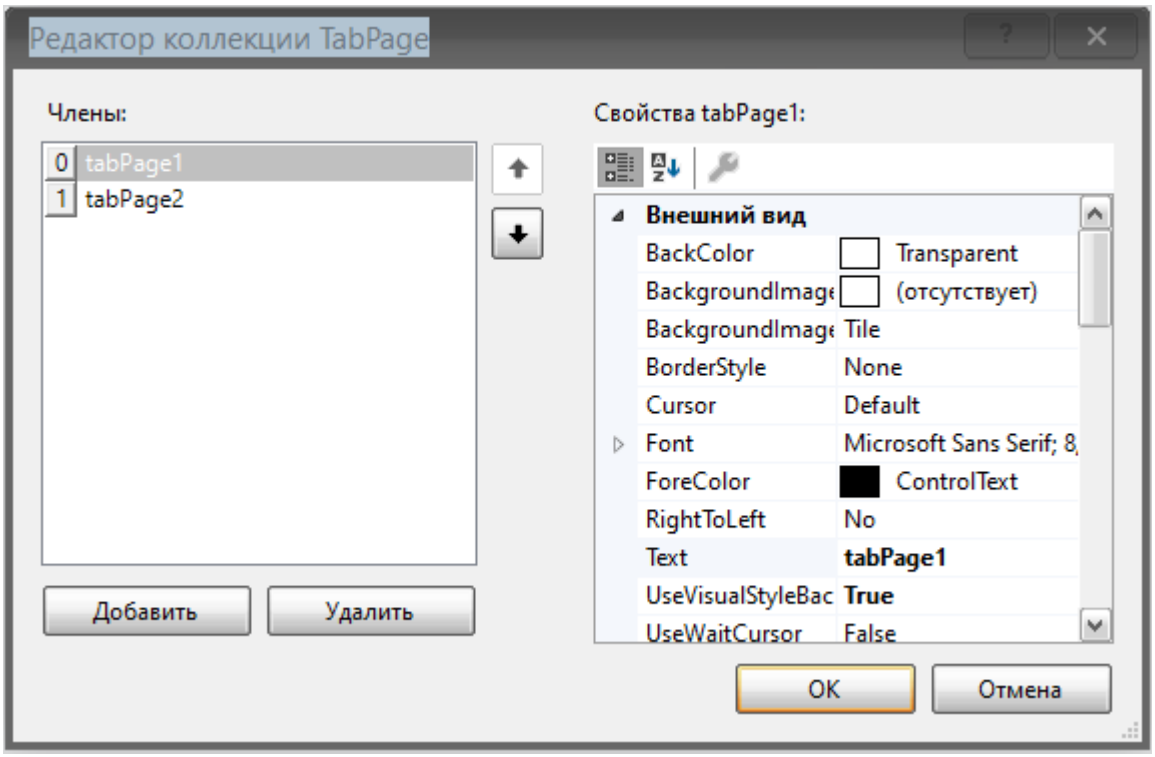

Рис. 14. Редактор колекції вкладок *TablPage*

Управління вкладками в коді

Для додавання нової вкладки потрібно її створити і додати в колекцію *tabControl1.TabPages* за допомогою методу *Add*:

```
//додавання вкладки
TabPage newTabPage = new TabPage();
newTabPage.Text = "Континенти";
tabControl1.TabPages.Add(newTabPage);
```

Видалення вкладки з набору здійснюється так:

```
// видалення вкладки
// за індексом
tabControl1.TabPages.RemoveAt(0);
// за об'єктом
tabControl1.TabPages.Remove(newTabPage);
```

Отримуючи в колекції *tabControl1.TabPages* потрібну вкладку за індексом, ми можемо легко маніпулювати нею:

```
// зміна властивостей
tabControl1.TabPages[0].Text = "Перша вкладка";
```

SplitContainer

Елемент *SplitContainer* дозволяє створювати дві розділені сплітером панелі (рис. 15). Змінюючи положення сплітера, можна змінити розміри цих панелей.

Використовуючи властивість *Orientation*, можна задати горизонтальне або вертикальне відображення сплітера (роздільної лінії) на форму. Ця властивість приймає значення *Horisontal* і *Vertical*.

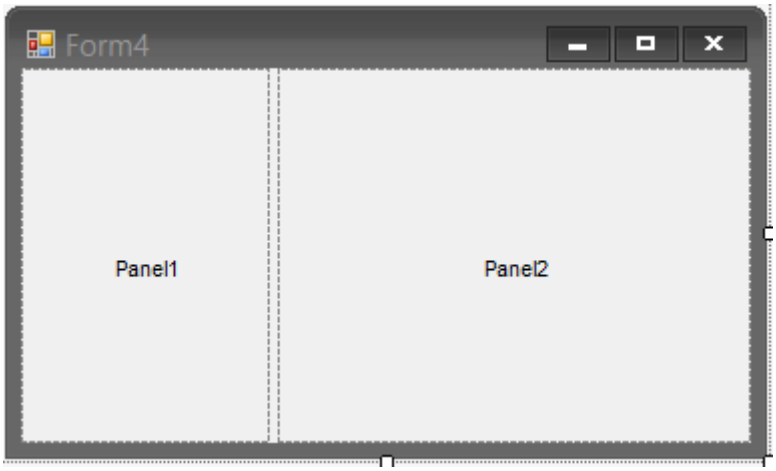

Рис. 15. Елемент *SplitContainer*

У разі, коли треба заборонити зміну положення сплітера, можна надати властивості *IsSplitterFixed* значення *true*. В цьому разі сплітер виявиться фіксованим, і ми не зможемо поміняти його положення.

За замовчуванням при розтягуванні форми або її звуженні також буде змінюватися розмір обох панелей цього контейнера. Однак ми можемо закріпити за однією панеллю фіксовану ширину (при вертикальній орієнтації сплітера) або висоту (при його горизонтальній орієнтації). Для цього треба встановити у елемента управління *SplitContainer* властивість *FixedPanel*. Як значення воно приймає панель, яку треба зафіксувати (у нашому прикладі це *None*, *Panel1*, *Panel2*).

Зміна положення сплітера в коді здійснюється властивістю *SplitterDistance*, яка задає положення сплітера в пікселях від лівого або верхнього краю елемен-

та *SplitContainer*. А за допомогою властивості *SplitterIncrement* можна задати крок, на який буде переміщатися сплітер при русі його за допомогою клавіш-стрілок.

Щоб приховати одну з двох панелей, ми можемо встановити властивість *Panel1Collapsed* або *Panel2Collapsed* в *true*.

Рекомендована література

Базова

1. Perkins B. ***Beginning C# 7 Programming with Visual Studio 2017*** / Benjamin Perkins, Jacob Vibe Hammer, Jon D. Reid. – Indianapolis, IN: Wrax, 2018. – 884 p. (Матеріали до книги – на github).
2. Шарп Д. ***Microsoft Visual C#. Подробное руководство***. 8-е изд. – СПб: Питер, 2017. – 848 с.
3. Johnson B. ***Professional Visual Studio*** (Programmer to Programmer). – John Wiley, 2017. – 832 p.
4. Murach J. ***Murach's C# 2015 (Training & Reference)*** / Anne Boehm, Joel Murach. – Fresno, CA: Mike Murach & Associates, 2016. – 883 p.
5. Ritchie P. ***Practical Microsoft Visual Studio 2015***. – Apress, 2016. – 199 p.
6. Olsson M. ***C# Quick Syntax Reference (The Expert's Voice)***. – Apress, 2013. – 122 p.
7. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5. – 6-е изд. –Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2013. — 1312 с.
8. Хейлсберг А. Язык программирования C#. Классика Computers Science. – 4-е изд. / А.Хейлсберг, М.Торгерсен, С.Вилтамут, П.Голд. – 2012. – СПб: Питер, 2012. – 784 с.
9. Єжова Л.Ф. ***Алгоритмізація і програмування процедур обробки інформації***: Навч.-метод. посібник. – К.: КНЕУ, 2000. – 152 с.

Допоміжна

1. Халецька З.П. Математична логіка та теорія алгоритмів: Навчальний посібник / З.П. Халецька, В.В. Нарадовий. – Кропивницький: РВВ КДПУ ім. В. Винниченка, 2017. – 128 с.
2. Пентус М. Р. Введение в математическую логику: Краткий конспект лекций (весна 2005, мехмат МГУ, 1-й курс, 2-й поток). – Предварительный вариант, 25.05.2005. – 87 с.
3. Завада О. П. Алгоритмізація і програмування: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2004. - 76 с.

12. Інформаційні ресурси

1. Документация по Visual Studio – Режим доступу: https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017
2. C# Tutorial and source code. – Режим доступу: http://csharp.net-informations.com
3. Язык программирования C# и .NET. – Режим доступу: https://metanit.com/sharp/general.php
4. Csharp Star. – Режим доступу: https://www.csharpstar.com
5. Алгоритмы. Алгоритмизация. Алгоритмические языки. – Режим доступу: http://book.kbsu.ru/theory/chapter7/1_7.html
6. Алгоритмы и алгоритмизация. – Режим доступу: http://smiuk.sfu-kras.ru/kodnyanko/site/algoritm/alg2.htm
7. The Visual Studio Blog. – Режим доступу: https://blogs.msdn.microsoft.com/visualstudio
8. .NET Tutorial - Hello World in 10 minutes. – Режим доступу: https://www.microsoft.com/net/learn/get-started-with-dotnet-tutorial
9. Microsoft Docs. – Режим доступу: https://docs.microsoft.com/ru-ru/?view=vs-2017