

## Лабораторна робота 3.

### Створення консольних застосунків

**Мета:** набути умінь і навичок роботи зі створення консольних програм на мові C#. Навчитись розробляти програми на мові C#, які мають лінійну, розгалужену та циклічну структури.

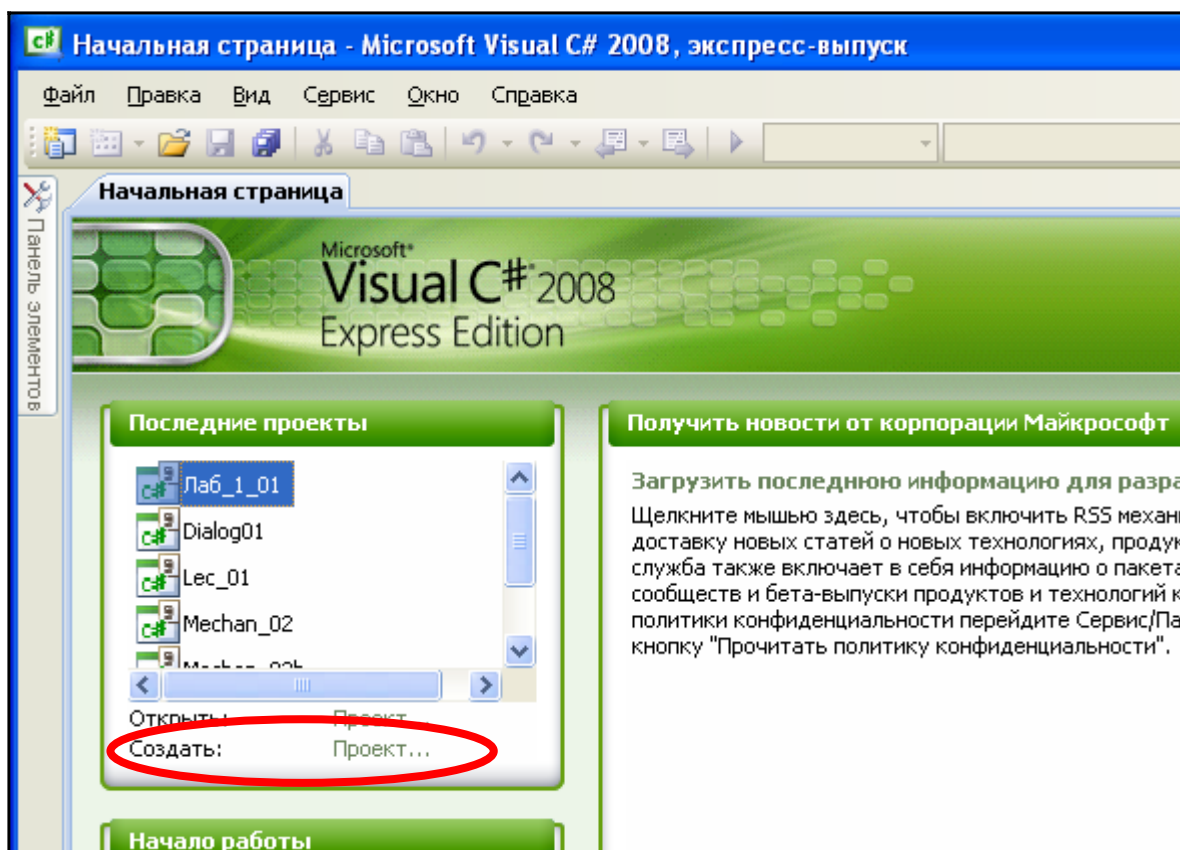
**Призначення:** ознайомлення з середовищем Visual Studio та основними конструкціями мови C#.

## Теоретичні відомості

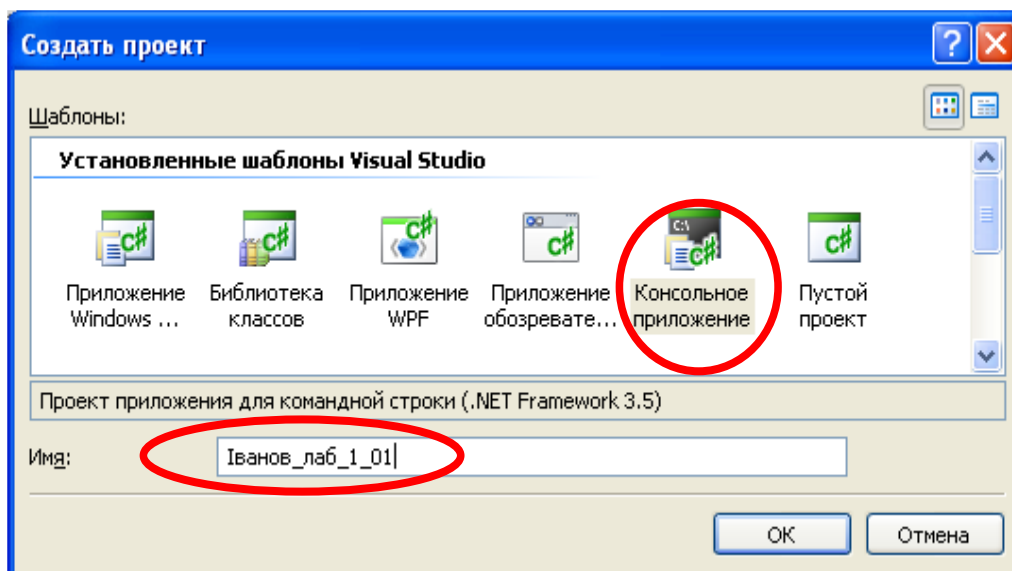
### 1.2. Створення консольного застосування

Запускаємо Microsoft Visual Studio.NET.

Для створення нового порожнього проекту C# в Visual Studio необхідно скористатися кнопкою Create Project (Создать Проект) на стартовій сторінці Microsoft Visual Studio (рис.1.1).

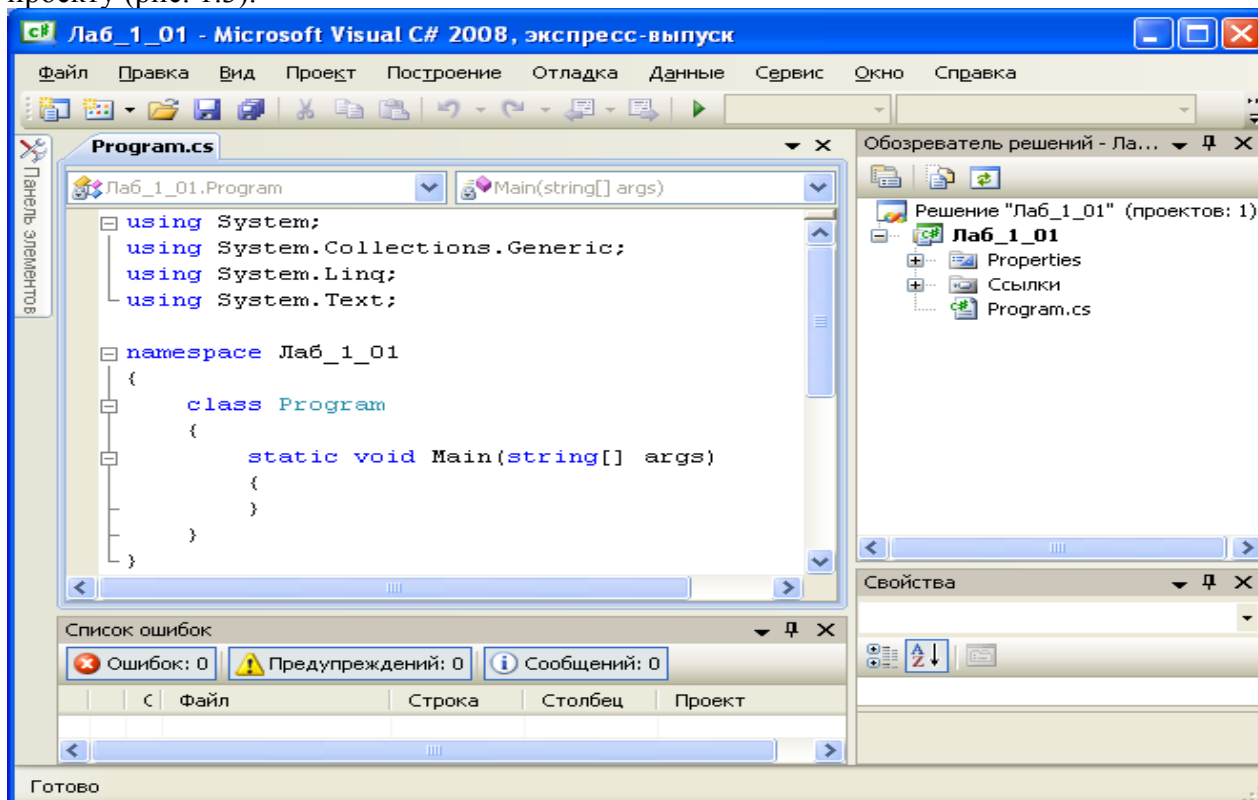


У вікні New Project, що з'явилося, ліворуч вибираємо Visual C#, а праворуч тип додатка - Console Application (рис. 1.2).



Як ім'я проекту (Name) наберіть „Своє прізвище знак підкреслення номер роботи”, наприклад **Иванов\_лаб\_1\_01** та натисніть на кнопку ОК для закриття даного діалогового вікна. За замовчуванням передбачено, що ім'я папки робочої області й ім'я проекту збігаються, що не обов'язково.

У результаті цього будуть створена папка робочої області **Иванов\_лаб\_1\_01**, а в цій папці ще одна папка проекту **Иванов\_лаб\_1\_01**, у якій будуть зберігатися файли проекту, і створений файл коду програми **Program.cs**. Крім цього, у папці робочої області створюється ще файл із ім'ям робочої області й розширенням **.sin**, що містить інформацію про налаштування робочої області, файли, папки і за допомогою якого відбувається завантаження робочого простору в середовище Visual Studio. У папці з назвою проекту створюються папки **bin**, **obj**, **Properties**. У папці **bin** знаходиться вкладена папка **Debug**, де згодом буде знаходитись виконуваний файл проекту **Иванов\_лаб\_1\_01.exe**. Папка **obj** використовується для зберігання інформації про об'єктні модулі, папка **Properties** — інформації про властивості проекту (рис. 1.3).

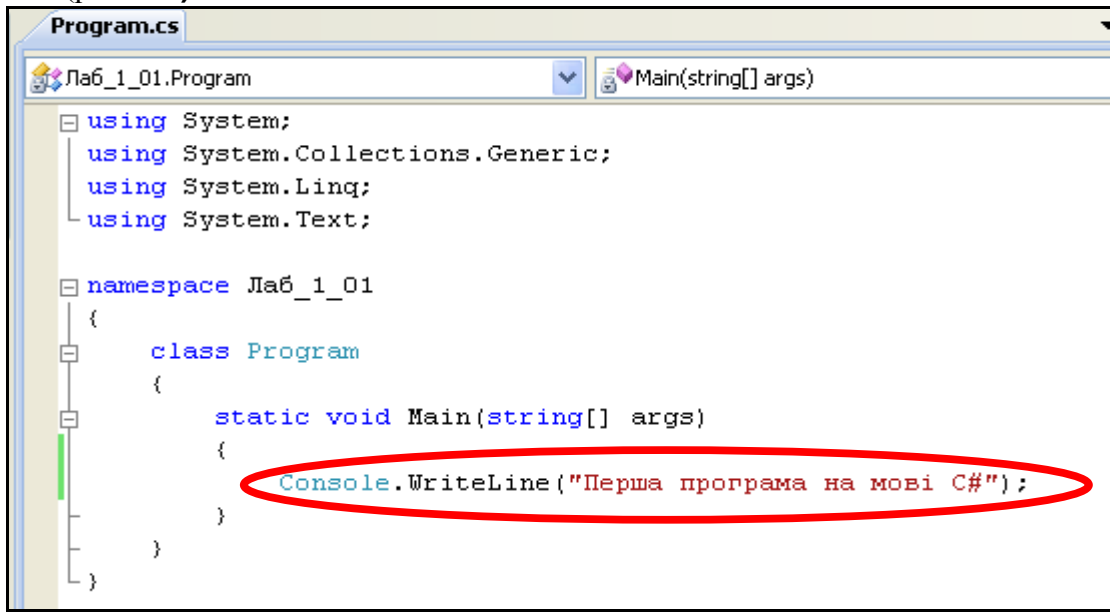


Додамо в код програми рядок, що виведе деяке повідомлення в консольне вікно.

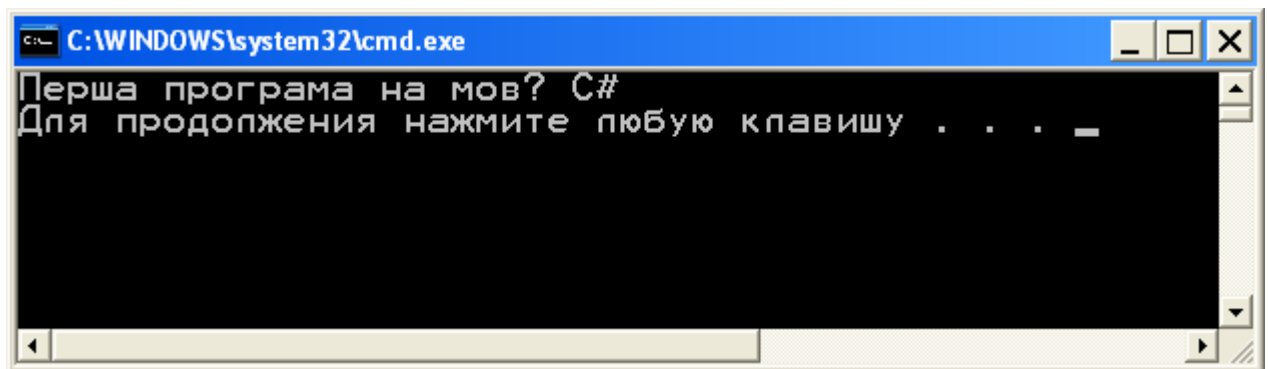
**Console.WriteLine("Перша програма на мові C#");**

Оскільки в програмі автоматично створений рядок **using System**, то замість довгих можна використовувати короткі імена методів, зокрема, замість **System.Console** можна писати просто **Console**, як записано в попередньому рядку.

Далі в програмі оголошений клас **Program**. У мові C# необхідно створити клас і у ньому функцію **Main** (функція **Main** обов'язково повинна бути в кожній програмі на C#, і саме із цієї функції й починається виконання програми). Ця функція пишеться з великої літери. C# розрізняє маленькі й великі літери. У функції **Main** виводимо на екран деякий рядок методом **WriteLine** (рис. 1.4).



Запускаємо програму, вибравши у верхньому меню робочої області **Debug - StartWithoutDebugging** або **Ctrl+F5**. Результат наведено на рис. 1.5.



Розширимо можливості програми, а саме додамо можливість зчитування даних з клавіатури й виконання найпростіших арифметичних операцій, знаходження суми, різниці, добутку й частки.

Для зчитування рядка символів, введеного із клавіатури в консольному вікні, використовується метод **Console.ReadLine()** простору імен **System**. Для перетворення рядка символів у число необхідно використати метод **Parse()**.

Використаємо відповідні методи в програмі:

```
float m; // Опис змінної m типу float
Console.WriteLine();
Console.WriteLine("Введіть ціле число: ");
// Вважаємо рядок символів методом Console.ReadLine() і за допомогою
```

```
// методу Parse() перетворимо його до цілого типу Int і присвоїмо
// змінній k значення цілого типу.
int k = Int32.Parse(Console.ReadLine());
Console.WriteLine("Було введено число - " + k);
k = k + k;
Console.WriteLine("Сума k+k =" + k);
k = k * k;
Console.WriteLine("Добуток k*k =" + k);
// Змінна I описана як float - дійсне число
float I = (float)k / ((float)k + (float)k);
Console.WriteLine("Вираз k / (k+k)= " + I);

Console.WriteLine();
Console.WriteLine("Введіть дробове число.");
Console.WriteLine("За розділювальний знак використовуйте кому: ");
// Зчитування рядка символів і перетворення його до типу float
m = float.Parse(Console.ReadLine());
m = (m + m) / (m * m);
Console.WriteLine("Вираз (m+m) / (m*m)= " + m);

Console.WriteLine();
Console.WriteLine("Для завершення натисніть ENTER");
Console.ReadLine();
```

Зверніть увагу на рядок:

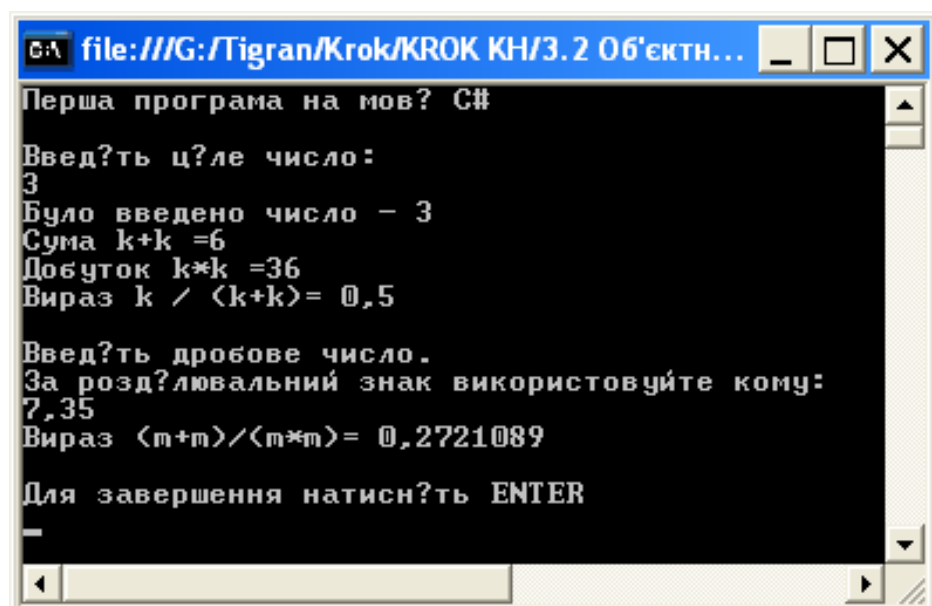
**float I = (float)k / ((float)k + (float)k);**

Тут у явному вигляді використовується приведення типу змінної **k** до дійсного типу **float**, інакше при проведенні обчислень результат від ділення буде приведений до цілого числа.

У рядку **m = float.Parse(Console.ReadLine());** використовується метод **Parse** для перетворення рядка символів у число дійсного типу **float**.

Результат перетворення присвоюється змінній **m**. Тип змінної описаний у рядку **float m; // Опис змінної m типу float**

Результат роботи програми наведено на рис. 1.6.



### 1.3. Основи C#

#### 1.3.1. Змінні мови C#

Для кожного типу даних C# існує відповідний тип даних у середовищі CLR (Common Language Runtime), що означає, що кожен тип має дві назви - повну (з CLR, його можна використовувати у будь-якій мові NET) і скорочену, яка використовується в C#. При розробці програм на C# можна використовувати як повний так і скорочений тип запису.

Отже, наступні три оголошення змінної **k** рівносильні:

**int k;**

**int32 k;**

**System.Int32 k;**

Аналогічно й з іншими типами мови C#

Основні типи даних та їх характеристики наведені у таблиці 1.1.

Таблиця 1.1 - Типи даних

Ім'я типу (C#)	Системний тип (CLR)	Значення та діапазон	Розмір і точність
<b>Логічний тип</b>			
bool	System.Boolean	true, false	8 біт
<b>Арифметичні цілочисельні типи</b>			
sbyte	System.Sbyte	- 128 ... 127	Знакове 8 Біт
byte	System.Byte	0...255	Беззнакове, 8 Біт
short	System.Short	-32768 ...32767	Знакове 16 Біт
ushort	System.Ushort	0... 65535	Беззнакове, 16 Біт
Int	System.Int32	- 2,147,483,648 ...2,147,483,647	Знакове, 32 Біт
uint	System.UInt32	0...4,294,967,295	Беззнакове, 32 Біт
long	System.Int64	-9,223,372,036,854,775.808 ... 9,223,372,036,854,775.807	Знакове, 64 Біт
ulong	System.UInt64	0... 18... 18,446,744,073,709,551, 615	Беззнакове, 64 Біт
<b>Арифметичний тип із плаваючою комою</b>			
float	System. Single	-3.402823e38 ... -г 3...402823e38	7 цифр
double	System.Double	- 1.79769313486232e308... 1.79769313486232e308	15-16 цифр
<b>Арифметичний тип з фіксованою комою</b>			
decimal	System. Decimal	-79,228,162,514,264,337,593, 543,950,335 79,228,162,514,264,337,593,543,950 ,335	28-29 значущих цифр
<b>Символьні типи</b>			
char	System.Char	U+0000-U+ffff	16 біт
string	System. String	Рядок із символів Unicode	Unicode символ
<b>Об'єктний тип</b>			
object	System.Object	Прабатько всіх вбудованих і користувацьких типів	

Оголошення змінної можна поєднати з ініціалізацією (завданням початкового значення):

**int z=88;**

Набір операторів для C# досить стандартний +, -, \/. Вони діють як і у будь-якій іншій мові. Відзначимо тільки, що / (ділення) стосовно цілих чисел дає цілу частину від ділення. Так, фрагмент

```
int k=100999, n=1000, s;
s=k/n;
Console.WriteLine(s.ToString());
```

виведе на екран 100, а не 101, тобто ніякого округлення не відбувається.

Є ще один оператор - %. Це - залишок від ділення. Наступний фрагмент виведе на екран 999:

```
int k=100999, n=1000, s;
s=k%n;
Console.WriteLine(s.ToString());
```

У C# існують оператори інкременту та декременту. Так, після наступного фрагмента k збільшиться на 1, а n-зменшиться на 1:

```
k++;
n--;
```

### 1.3.2. Оператори мови C#

**Оператор присвоювання.** У C# присвоювання формально вважається операцією. Запис  $X = \text{expr};$  вважають оператором присвоювання, так само, як і одночасне присвоювання зі списком змінних у лівій частині:

$X1 = X2 = \dots = \text{expr};$

**Блок або складений оператор.** За допомогою фігурних дужок кілька операторів можна об'єднати в єдину синтаксичну конструкцію, яку називають блоком або складеним оператором:

```
{
оператор_1
...
оператор_N
}
```

Синтаксично блок сприймається як одиничний оператор і може використовуватися всюди в конструкціях, де синтаксис вимагає одного оператора. Тіло циклу, гілки оператора **if**, як правило, представляються блоком.

### 1. Логічні оператори

Логічні оператори, що існують у C#, наведені у таблиці 1.2.

**Таблиця 1.2**

Оператор	Опис	Приклад
&&	Логічне ТА. Результат дорівнює true, тільки якщо обидва операнди дорівнюють true	$(x \leq 8) \ \&\& \ (y = 5)$
!!	Логічне АБО. Результат дорівнює false, тільки якщо обидва операнди дорівнюють false	$(y > 8) \ \&\& \ (y < 5)$
!	Заперечення. Змінює логічне значення на протилежне	$\text{if}(!(a = 1))\dots$

Всі ці оператори повертають результат типу **bool**.

Зверніть увагу, що для логічного *дорівнює* (тобто для відповіді на питання "чи вірно, що щось рівне чомусь") використовується знак подвійної рівності (==). Знак одинарної рівності (=) використовується для *присвоювання*. Для знака == існує парний знак != ("не дорівнює"). Так, наведений вище приклад для оператора ! можна переписати так:

```
if(a!=b)...
```

У С# не можна замість **false** використовувати 0, а замість **true** -будь-яке ненульове число. Так, наступний фрагмент містить помилку:

```
int k;  
...  
if(k) // Помилка!  
...
```

## 2. Оператори вибору

У мові С# для вибору однієї з декількох можливостей використовуються дві конструкції - *if* і *switch*. Першу з них зазвичай називають альтернативним вибором, другу - розбором випадків.

*Оператор if* Синтаксис оператора И:

```
If (вираз_1) оператор_1  
else if (вираз_2) оператор 2  
...  
else if (вираз_K) оператор_K  
else оператор_N
```

Вирази *if* повинні вкладатися в круглі дужки і бути булевого типу (вирази повинні давати значення **true** або **false**). Арифметичний тип не має явних або неявних перетворень до булевого типу. За правилами синтаксису мови, *then*-гілка оператора слідує відразу за круглою дужкою без ключового слова **then**, типового для більшості мов програмування. Кожний з операторів може бути блоком - зокрема, *if*-оператором. Тому можлива й така конструкція:

```
If (вираз1) if (вираз2) if (вираз3)...
```

Гілки **else** та *if*, що дозволяють організувати вибір з багатьох можливостей, можуть бути відсутніми. Може бути опущена й заключна **else**-гілка. У цьому випадку коротка форма оператора **if** задає альтернативний вибір - робити або не робити - виконувати або не виконувати *then*-оператор.

Вирази *if* перевіряються в порядку їх написання. Як тільки отримане значення **true**, перевірка припиняється й виконується оператор (це може бути блок), який слідує за виразом, що одержав значення **true**. Із завершенням цього оператора завершується й оператор *if*. Гілка **else**, якщо вона є, відноситься до найближчого відкритого *if*.

Оператор *if* слугує для розгалуження програми на два напрямки. Якщо деяка умова виконується, то програма йде в одну сторону, якщо не виконується - в іншу.

*Приклад*, який визначає, парне чи непарне число ввів користувач:

```
class Program  
{  
    static void Main(string[] args)  
    {  
        // Читаємо символ, введений з клавіатури (метод ReadLine()).  
        // за допомогою методу Parse перетворимо його у ціле число  
        // й присвоюємо значення змінній k  
        Console.WriteLine("Введіть ціле число");
```

```

        int k1 = Int32.Parse(Console.ReadLine());
        int b = k1/2;
        if(2*b == k1)
        { Console.WriteLine("Парне число"); }
        else
        { Console.WriteLine("Непарне число");}

        Console.WriteLine("Для завершення натисніть ENTER");
        Console.ReadLine();
    }
}

```

Фігурні дужки можна не писати у випадку одного оператора. Гілка *else* теж не є обов'язковою - все залежить від конкретної задачі.

**Оператор switch.** Важливим випадком вибору з декількох варіантів є ситуація, при якій вибір варіанта визначається значеннями деякого виразу. Відповідний оператор C# називається оператором *switch*. Його синтаксис:

```

switch(вираз)
{
    case константний_вираз_1: [оператори_1
    оператор_переходу_1]
    ...
    case константний_вираз_K: [оператори_K
    оператор_переходу_K]
    [default: оператори_N оператор_переходу_N]
}

```

Гілка *default* може бути відсутньою. За синтаксисом припустимо, щоб після двокрапки слідувала порожня послідовність операторів, а не послідовність, яка закінчується оператором переходу. Константні вирази в *case* повинні мати той же тип, що й switch-вираз.

Спочатку обчислюється значення switch-виразу. Потім воно по черзі в порядку проходження *case* порівнюється на збіг з константними виразами. Як тільки досягнутий збіг, виконується відповідна послідовність операторів case-гілки. Оскільки останній оператор цієї послідовності є оператором переходу (найчастіше це оператор *break*), то звичайно він завершує виконання оператора *switch*. Case-гілка може бути порожньою послідовністю операторів. Тоді, у випадку збігу константного виразу цієї гілки зі значенням switch-виразу, буде виконуватися перша непушта послідовність чергової case-гілки. Якщо значення switch-виразу не збігається з жодним константним виразом, то виконується послідовність операторів гілки *default*, якщо ж такої гілки немає, то оператор *switch* еквівалентний порожньому оператору.

#### Приклад

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Введіть оцінку");
        int k2 = Int32.Parse(Console.ReadLine());
        Console.WriteLine();

        Console.WriteLine("Введено оцінку:" + k2.ToString());

        switch (k2)
        {
            case 1:
            case 2: Console.WriteLine("    Незадовільно"); break;

```



```

        case 3: Console.WriteLine("    Задовільно"); break;
        case 4: Console.WriteLine("    Добре"); break;
        case 5: Console.WriteLine("    Відмінно"); break;
        default:
            Console.WriteLine("    Помилка"); break;
    }

    Console.WriteLine("Для завершення натисніть ENTER");
    Console.ReadLine();
}
}

```

У наведеному прикладі залежно від введеного користувачем числа на екран виводиться та або інша оцінка. Якщо число **k** не лежить у проміжку від 1 до 5, то виконуються оператори у гілці **default** і виводиться напис **Помилка**. Гілка **default** необов'язкова. Зверніть увагу на оператор **break**. Якщо його не написати, то будуть виконуватися оператори з наступної гілки **case** до рядка з оператором **break**. Якщо в деякій гілці **case** або **default** є оператори, то написання **break** обов'язково. Так, у наступних двох ділянках коду програми є помилки:

```

...
case 1:
    Console.WriteLine("Зовсім незадовільно");
    // Помилка! Тут пропущений break
case 2:
    Console.WriteLine("Незадовільно "); ]
    break;
...
default:
    Console.WriteLine("...");
    // Помилка! Тут пропущений break
}

```

Застосовуючи оператор **switch**, необхідно закінчувати кожну case-гілку оператором **break**.

### 3. Оператори циклу

**Оператор for.** Оператор циклу **for** узагальнює відому конструкцію циклу типу арифметичної професії. Його синтаксис:

#### **for(ініціалізатори; умова; список\_виразів) оператор**

Оператор, що знаходиться після закриваючої дужки, задає тіло циклу. У більшості випадків тілом циклу є блок. Скільки разів буде виконуватися тіло циклу, залежить від трьох керуючих елементів, заданих у дужках. **Ініціалізатори** задають початкове значення однієї або декількох змінних, які називають лічильниками або просто змінними циклу. У більшості випадків цикл **for** має один лічильник. **Умова** задає умову закінчення циклу, відповідний вираз при обчисленні повинен одержувати значення **true** або **false**. **Список виразів**, записаний через кому, показує, як змінюються лічильники циклу на кожному кроці виконання. Якщо умова циклу істинна, то виконується тіло циклу, потім змінюються значення лічильників і знову перевіряється умова. Як тільки умова стає помилковою, цикл завершує свою роботу. У циклі **for** тіло циклу може жодного разу не виконуватися, якщо умова циклу хибна після ініціалізації, а може відбуватися зациклення, якщо умова завжди залишається істинною. У нормальній ситуації тіло циклу виконується скінчену кількість разів.

Лічильники циклу найчастіше оголошуються безпосередньо в ініціалізаторі й відповідно є змінними, локалізованими в циклі, і після завершення циклу вони перестають існувати.

У тих вішалках, коли передбачається можливість передчасного завершення циклу з допомогою одного з операторів переходу, лічильники оголошуються до початку циклу, що дозволяє аналізувати їхні значення при виході із циклу.

#### **Приклад**

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Введіть число");
        int k3 = Int32.Parse(Console.ReadLine());
        int sum = 0;
        for (int i = 1; i <= k3; i++)
        {
            sum += i;          // Еквівалентно sum=sum+i
        }
        Console.WriteLine(sum);
        Console.WriteLine("Для завершення натисніть ENTER");
        Console.ReadLine();
    }
}
```

Цей приклад підраховує суму чисел від 1 до введеного користувачем числа k. Сума записується в змінну sum і виводиться на екран.

**Цикли while.** Цикл **while(вираз)** є універсальним видом циклу, що є в усіх мовах програмування. Тіло циклу виконується доти, поки залишається істинним вираз **while**. У мові C# у цього виду циклу дві модифікації - з перевіркою умови на початку й наприкінці циклу. Перша модифікація має наступний синтаксис:

#### **while(вираз) оператор**

Ця модифікація відповідає алгоритму: "спочатку перевір, а потім роби". У результаті перевірки може виявитися, що й робити нічого не потрібно. Тіло такого циклу може жодного разу не виконуватися. Кожне виконання тіла циклу - це черговий крок до завершення циклу.

Цикл, що перевіряє умову завершення наприкінці, відповідає алгоритму: "спочатку роби, а потім перевір". Тіло такого циклу виконується, щонайменше, один раз. Синтаксис цієї модифікації:

```
do
    оператор
while(вираз);
```

Обидва ці цикли використовуються, як правило, тоді, коли точно не відомо, скільки разів цикл повинен виконатися. Наприклад, при введенні користувачем пароля або при підрахунку чого-небудь із певною точністю. Ці цикли будуть виконуватися доти, поки умова в круглих дужках після слова while буде істинною. Як тільки умова стане рівною false, виконання циклу припиняється. Найважливіша відмінність між while й do-while у тому, що while може не виконатися жодного разу, тоді як do-while принаймні один раз виконається.

#### **Приклад**

```
class Program
{
    static void Main(string[] args)
    {
```

```

        string password;
        do
        {
            Console.WriteLine("Введіть пароль");
            password = Console.ReadLine();
        }
        while (password != "C#");
        Console.WriteLine("Для завершення натисніть ENTER");
        Console.ReadLine();
    }
}

```

У прикладі цикл буде виконуватися доти, поки користувач не введе правильний пароль (C#).

#### 4. Оператори переходу

Операторів переходу, що дозволяють перервати природний порядок виконання операторів блоку, у мові C# є декілька.

**Оператори break й continue** дозволяють при виконанні деякої умови вийти із циклу, з оператора вибору, із блоку.

Оператор **break** може стояти в тілі циклу або завершувати case-гілку в операторі **switch**. Приклад його використання в операторі switch уже демонструвався. При виконанні оператора **break** у тілі циклу завершується виконання самого внутрішнього циклу. У тілі циклу, найчастіше, оператор **break** міститься в одній з гілок оператора **if**, що перевіряє умову передчасного завершення циклу:

```

class Program
{
    static void Main(string[] args)
    {
        int i2, j2;
        for(i2 = 1; i2<100; i2++)
        {
            for(j2=1; j2<10; j2++)
            {
                if (j2>=3) break;
            }
            Console.WriteLine(" Вихід із циклу j при j = {0}", j2);
            if (i2>=3) break;
        }
        Console.WriteLine(" Вихід із циклу i при i= {0}", i2);
    }
}

```

Оператор **continue** використовується тільки в тілі циклу. На відміну від оператора **break**, що завершує внутрішній цикл, **continue** здійснює перехід до наступної ітерації цього циклу.

**Оператор return.** Ще одним оператором, що відноситься до групи операторів переходу, є оператор return, який дозволяє завершити виконання процедури або функції. Його синтаксис:

**return [вираз];**

Для функцій його присутність і аргумент обов'язковий, оскільки вираз в операторі return задає значення, що повертається функцією.

#### 1.3.3. Клас Math

Стандартні математичні функції в C# зібрані в окремому класі

**Math** й є методами класу.

Клас **Math**, що містить стандартні математичні функції, містить два статичні поля, константи, що задають е та т. а також 23 статичних методи. Методи задають:

- тригонометричні функції - Sin, Cos, Tan;
- зворотні тригонометричні функції - ASin, ACos, ATan, ATan2

(sinx, cosx);

- гіперболічні функції - Tanh, Sinh, Cosh;
- експоненту й логарифмічні функції - Exp, Log, Log10;
- модуль, корінь, знак - Abs, Sqrt, Sign;
- функції округлення - Ceiling, Floor, Round;
- мінімум, максимум, степінь, залишок - Min, Max, Pow, IEEERemainder.

У наступному прикладі користувач визначає, яку функцію він хоче обчислити й при яких значеннях її параметрів.

#### **Приклад**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Лаб_1_01a
{
    class Program
    {
        static void Main(string[] args)
        {
            double a, b, t, t0, dt, y;
            string NameFunction;
            Console.WriteLine("Введіть ім'я F(t) досліджуваної функції a*F(b*t) "
                + " (sin, cos, tan, cosh, ln, tanh)");
            NameFunction = Console.ReadLine();
            Console.WriteLine("Введіть параметр a (double)");
            a = double.Parse(Console.ReadLine());
            Console.WriteLine("Введіть параметр b (double)");
            b = double.Parse(Console.ReadLine());
            Console.WriteLine("Введіть початковий час t0 (double)");
            t0 = double.Parse(Console.ReadLine());
            const int points = 10;
            dt = 0.2;
            t = t0;
            for(int i = 1; i <= points; i++)
            {
                t = t + a*dt;
                switch (NameFunction)
                {
                    case ("sin"):
                        y = a*Math.Sin(b*t);
                        break;
                    case ("cos"):
                        y = a*Math.Cos(b*t);
                        break;
                    case ("tan"):
                        y = a*Math.Tan(b*t);
                        break;
                    case ("cosh"):
                        y = a*Math.Cosh(b*t);
```

```

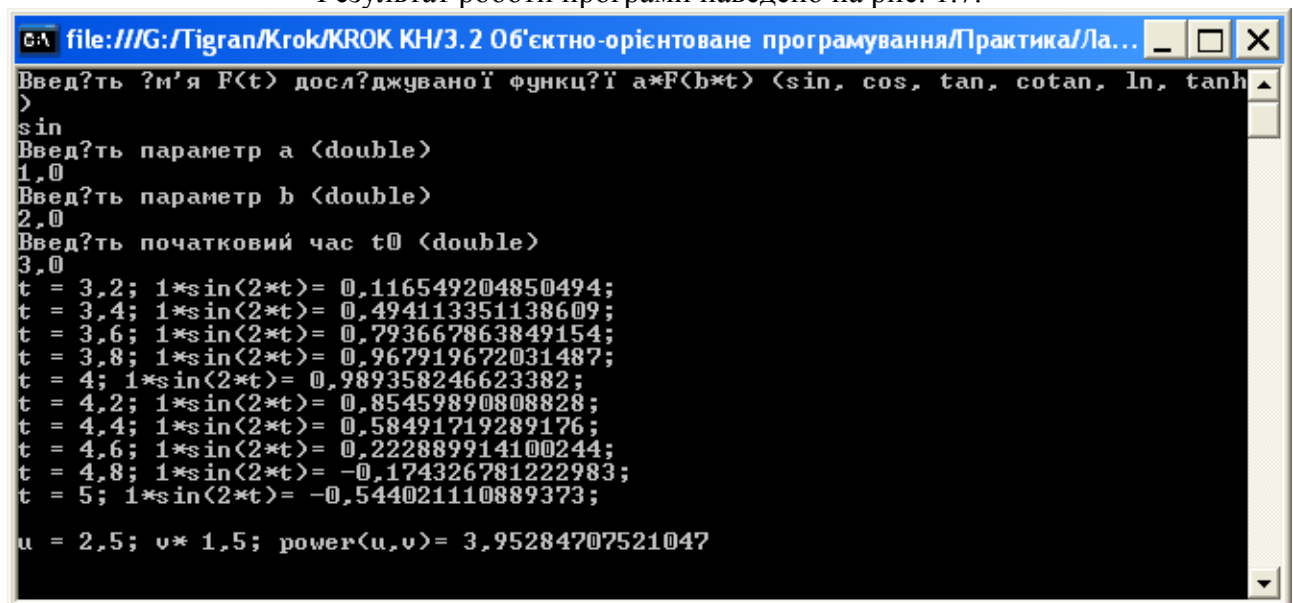
        break;
    case ("ln"):
        y = a*Math.Log(b*t);
        break;
    case ("tanh"):
        y = a*Math.Tanh(b*t);
        break;
    default:
        y=1;
        break;
    }
    Console.WriteLine ("t = " + t + "; " + a + "*" +
        NameFunction + "(" + b + "*" + t) = " + y + ";");
}
Console.WriteLine();
double u = 2.5, v = 1.5, p;
p= Math.Pow(u,v);
Console.WriteLine ("u = " + u + "; v* " + v + "; power(u,v)= " + p);

Console.ReadLine ();

}
}
}

```

Результат роботи програми наведено на рис. 1.7.



```

file:///G:/Tigran/Krok/KROK KH/3.2 Об'єктно-орієнтоване програмування/Практика/Ла...
Введ?ть ?м'я F(t) досл?джуваної функц?ї a*F(b*t) <sin, cos, tan, cotan, ln, tanh>
>
sin
Введ?ть параметр a <double>
1,0
Введ?ть параметр b <double>
2,0
Введ?ть початковий час t0 <double>
3,0
t = 3,2; 1*sin(2*t)= 0,116549204850494;
t = 3,4; 1*sin(2*t)= 0,494113351138609;
t = 3,6; 1*sin(2*t)= 0,793667863849154;
t = 3,8; 1*sin(2*t)= 0,967919672031487;
t = 4; 1*sin(2*t)= 0,989358246623382;
t = 4,2; 1*sin(2*t)= 0,85459890808828;
t = 4,4; 1*sin(2*t)= 0,58491719289176;
t = 4,6; 1*sin(2*t)= 0,222889914100244;
t = 4,8; 1*sin(2*t)= -0,174326781222983;
t = 5; 1*sin(2*t)= -0,544021110889373;

u = 2,5; v* 1,5; power(u,v)= 3,95284707521047

```