Лабораторна робота № 3
Програмування обчислювальних процесів, що розгалужуються

Мета роботи – набуття практичних навичок з підготовки, налагодження і виконання програм, що розгалужуються.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

методики розробки програми із загальною лінійною частиною і кількома гілками;

алгоритмів виконання та синтаксис операторів if, if / else, switch і умовного виразу (?);

уміння:

складати програми з розгалуженнями;

виконувати налагодження та покрокове тестування програми з розгалуженнями в середовищі системи Visual C# .NET;

комунікації:

рекомендації команді учасників проекту щодо доцільності застосування поточного сценарію у вигляді алгоритмічних структур із розгалуженнями;

робота в команді над окремими частинами складного коду, який складається із структур вибору;

автономність і відповідальність:

прийняття рішення щодо розподілу початкового коду складної програми, в яку входять структури з розгалуженнями;

самостійний обґрунтування можливих варіантів C# ─ реалізацій структур вибору.

Основні положення

У лінійних програмах всі оператори виконувалися послідовно і, як наслідок, вони не здатні реагувати на поточні умови.

Однак часто в процесі реалізації поточного сценарію потрібно змінювати потік керування, реагуючи на якісь зовнішні події.

Потік керування становить порядок, в якому виконуються оператори програми. Крім того, часто використовуються терміни «порядок виконання» і «керуючий потік».

Гілкою називають сегмент програми, що містить один оператор або їх групу. Оператор розгалуження дозволяє запускати потрібний блок операторів. Вибір здійснюється за умовою. Оператори розгалуження часто називають операторами вибору.

C# забезпечує три типи структур вибору альтернатив:

єдиний вибір – структура if (ЯКЩО);

подвійний вибір – структура if / else (ЯКЩО / ІНАКШЕ);

множинний вибір – структура switch.

Структура вибору if.

Синтаксис оператора if відображений у такому синтаксичному блоці:

Оператор_if ::=

if(<Умова>)

<Оператор>;

Обрана тут форма запису – це спрощена версія часто використовуваної для опису синтаксису комп'ютерної мови нотації, ку називають формою Бекуса - Наура (Backus - Naur), або BNF. Вона була розроблена Дж. Бекусом і П. Науром.

Форма запису синтаксису складається з таких елементів:

Символ :: =, що означає "визначається як".

Метазмінні (розміщені в кутових дужках) в формі <Слово>.

Символ, що складається із двох квадратних дужок [ ] та позначає необов'язкові елементи [<це необов'язково>].

Три крапки ... які вказують на необмежену кількість елементів.

Вертикальна риса |, що вказує на можливі альтернативи.

Вираження логічного типу *(<Умова>)* завжди дає одне з двох значень: true (істина) або false (неправда).

<Оператор>, що слідує за логічним виразом, виконується лише в тому випадку, якщо останнє істинно.

Графічна схема оператора наведена на рис. 36.

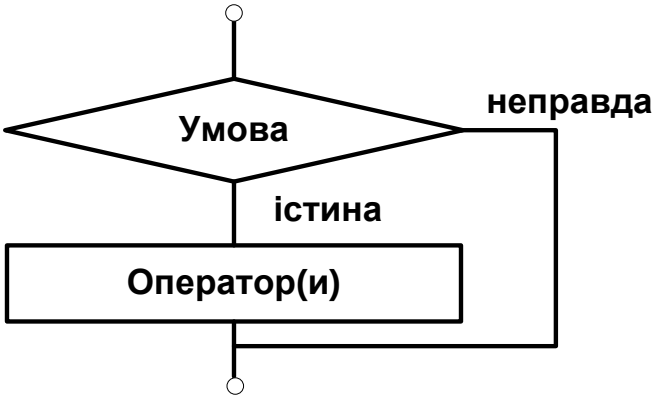


Рис. 36. **Графічна схема оператора if**

Приклад 1. Перевірка правильності введення змінної, яка може містити числа від 1 до 31.

```
using System;
        class Class1
        {
                static void Main( )
                {
                        int valor;
                        Console.WriteLine("Введіть число місяця");
                        valor = Convert.ToInt32(Console.ReadLine());
                        if (valor < 1  ||  valor >31)
                                Console.WriteLine("Помилка введення!");
                        Console.WriteLine("Ви ввели число, рівне {0}",
valor);
                }
        }
```

Як оператори не можна використовувати оголошення і визначення. Однак тут можуть бути складені оператори і блоки:

```
Складений_оператор ::=
                        {
                        <Оператори>
                        }
```

Структура вибору if / else.

Синтаксичний блок оператора if / else:

```
Оператор  if / else ::=
```

if (<Умова>)
<Оператор_1>; | <Складений_оператор_1>
else
<Оператор_2>; | <Складений_оператор_2>

<Оператор_1>; | <Складений_оператор_1> виконується лише в тому випадку, коли логічний вираз (<Умова>) дорівнює true.

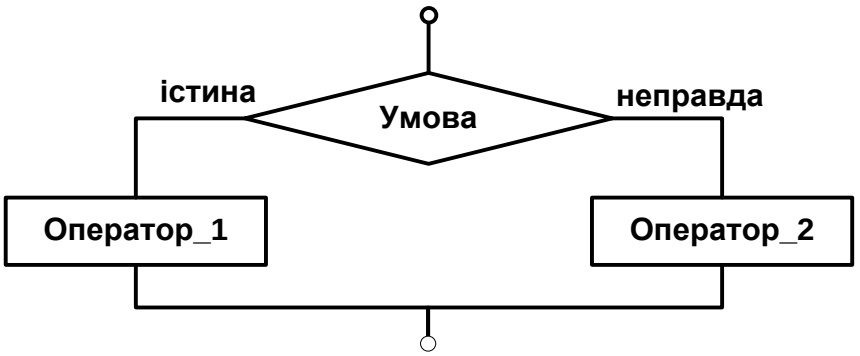
<Оператор_2>; | <Складений_оператор_2> виконується лише тоді, коли (<Умова>) дорівнює false.

Перед else обов'язково ставиться крапка з комою.

Графічна схема оператора наведена на рис. 37.

Приклад 2. Знайти мінімум із двох чисел.

using System;
class Class1
{




```
...
if (Умова)
  Оператор_1;
else
  Оператор_2;
...
```

Рис. 37. **Графічна схема оператора if / else**

```
static void Main()
{
    int x, y, min;
    Console.WriteLine("Послідовно введіть два цілих числа");
    x = Convert.ToInt32(Console.ReadLine());
    y = Convert.ToInt32(Console.ReadLine());
    if  (x<y)
        min = x;
    else
        min = y;
```

```
                Console.WriteLine("Мінімальне число дорівнює {0}", min);
            }
        }
```

Оператори if можуть бути вкладені один в один.

Приклад 3. Знайти максимальне число із трьох чисел a, b, c.

```
using System;
    class Class1
    {
        static void Main()
        {

            int a, b, c, max;
            Console.WriteLine("Послідовно введіть три цілих числа");
            a = Convert.ToInt32(Console.ReadLine());
            b = Convert.ToInt32(Console.ReadLine());
            c = Convert.ToInt32(Console.ReadLine());
            if  (a > b  &&  a > c)
                max = a;
            else
                if  (b > c)
                max = b;
            else
                max = c;
            Console.WriteLine("Максимальне число дорівнює {0}", max);
        }
    }
```

Обидві гілки повного умовного оператора можуть бути складеними.

Множинний вибір – структура switch

Оператор switch дозволяє програмі обрати одну з кількох дій на основі значення заданого виразу. Логіка, яка реалізована switch, подібна до логіки оператора if / else.

Синтаксичний блок оператора switch:

switch (<Вираз_switch >)

{
case <Константний_вираз>:
[<Оператор>;
<Оператор>; <Оператор>;
<Оператор_break>; | <Оператор_goto>]
case <Константний_вираз>:
[<Оператор>;
<Оператор>; <Оператор>;
<Оператор_break>; | <Оператор_goto>]
<Будь-яка кількість блоків case>
[default:
[<Оператор>;
<Оператор>; <Оператор>;
<Оператор_break>; | <Оператор_goto>]
}

Графічна схема оператора switch наведена на рис. 38.

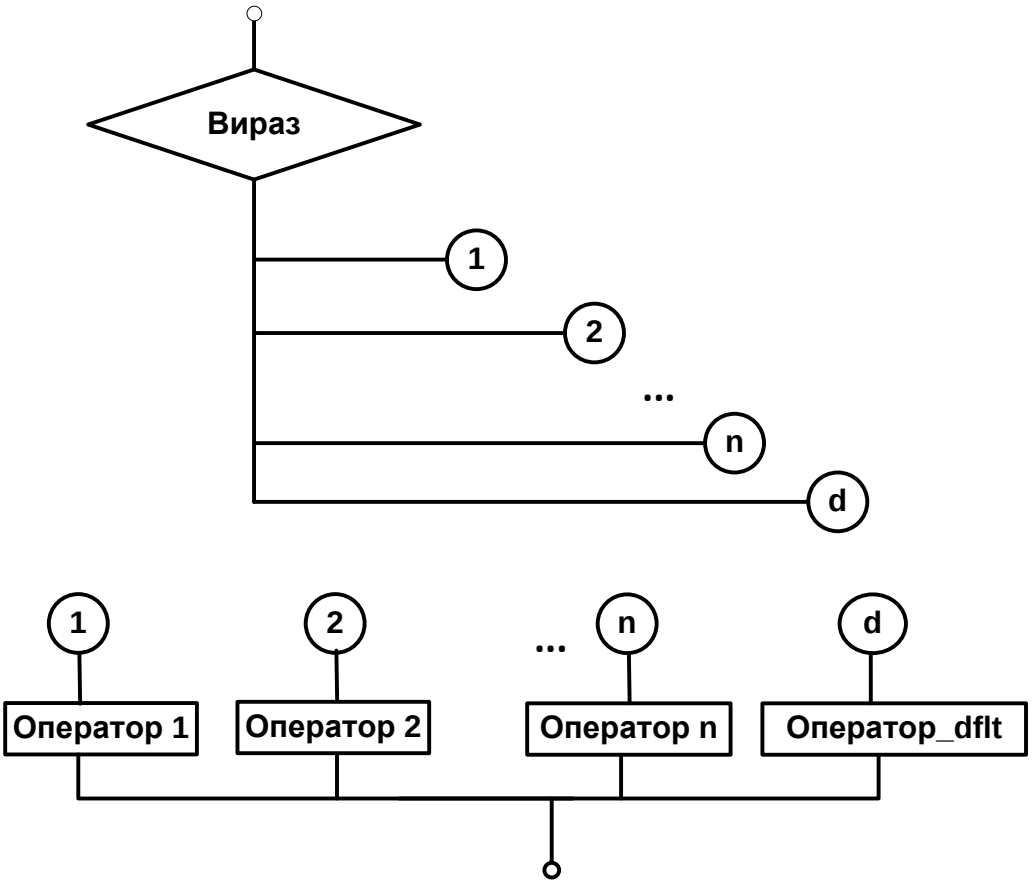


Рис. 38. **Графічна схема оператора switch**

Може бути один або не бути жодного блоку default.

Термін (<Вираз_switch>), використовуваний разом із ключовим словом switch, – це загальноприйнята назва керуючого виразу.

Розділи case і default називають розділами вибору.

<Константний_вираз>, що йде услід за ключовим словом case, називають значенням або case-міткою. Кожна з них повинна бути унікальною.

Найпоширеніший спосіб завершення розділу вибору – застосування <Оператор_break> або <Оператор_goto>.

Хоча розділи case і default можна розміщувати в будь-якій послідовності, гарний стиль програмування передбачає, що розділ default розміщується наприкінці оператора switch.

Коли потік керування переходить від одного розділу switch до іншого, таке виконання називається провалом. Для його запобігання застосовується оператор break або goto.

Приклад 4. Необхідно проаналізувати значення змінної nota, що є виставленою оцінкою.

```
using System;
class Nota
{
public static void Main()
{
    int nota;

    Console.WriteLine("Ваша оцінка (від 2 до 5)? ");
    nota = Convert.ToInt32(Console.ReadLine());

    switch(nota)
    {
            case 2:
                Console.WriteLine("Ви вибрали 2 ");
                Console.WriteLine("Оцінка - незадовільно");
                break;
            case 3:
                Console.WriteLine("Ви вибрали 3 ");
                Console.WriteLine("Оцінка - задовільно");
                break;
```

```
                case 4:
                        Console.WriteLine("Ви вибрали 4 ");
                        Console.WriteLine("Оцінка - добре");
                        break;
                case 5:
                        Console.WriteLine("Ви вибрали 5 ");
                        Console.WriteLine("Оцінка - відмінно");
                        break;
                default:
                        Console.WriteLine("Помилка вибору. Ви повинні
вибрати число " +
                        "між 2 and 5");
                        break;
            }
        }
        }
```

Приклад 5. Створення простого меню.

```
        using System;
                class Class1
                {
                        static void Main()
                        {
                                char vibor;
                                Console.Write("МЕНЮ:\t A(dd) D(elete) S(ort) Q(uit)
\n");
                                Console.Write("Ваш вибір? ");
                                vibor = Convert.ToChar(Console.Read());
                                switch (Char.ToUpper(vibor))
                                {
                                        case 'A':
                                                Console.Write("Обрано Add\n");
                                                break;
                                        case 'D':
                                                Console.Write("Обрано Delete\n");
                                                break;
                                        case 'S':
                                                Console.Write("Обрано Sort\n");
```

```
                    break;
                case 'Q':
                    Console.Write("Обрано Quit\n");
                    break;
                default:
                    Console.Write("Введений помилковий символ\n");
                    break;
            }
        Console.Write("\nДля завершення програми натисніть <Enter>");
        Console.ReadLine();  // для паузи
        Console.ReadLine();  // для паузи
        }
    }
```

Умовний вираз.

У C# є ще одна скорочена форма умовного оператора – так званий умовний вираз. Його синтаксис:

<Логічний_вираз_1> ? <Вираз_2> : <Вираз_3>;

Сенс цієї конструкції полягає в такому:

обчислюється <Логічний_вираз_1>; якщо він істинний (true), то результатом буде <Вираз_2>, у протилежному випадку результатом буде <Вираз_3>.

По суті, умовний вираз еквівалентний такому умовному операторові:

```
if  ( Логічний_вираз_1 )
        Вираз_2;
else
        Вираз_3;
```

Приклад. Знайти максимум із двох чисел.

```
…
int x = 13, y = 7, max;
max  =  (x > y) ? x : y;
Console.WriteLine("max = {0}", max);
…
```

Порядок виконання лабораторної роботи

Загальна частина.

1. Набрати, відкомпілювати і запустити на виконання приклади програм, які були наведені в розділі «Основні положення» даної лабораторної роботи.

2. Проекспериментувати з програмами:

змінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина.

1. Формули для обчислення у вигляді F = f (a, b, c, x) і опис змінних взяти з відповідного варіанта індивідуального завдання у викладача (див. додатковий файл з індивідуальними завданнями).

Завдання відповідає такому сценарію роботи:

Введіть чотири дійсні числа:
a =? <число_1> <Введення>
b =? < число _1> <Введення>
c =? < число _1> <Введення>
x =? < число _1> <Введення>
F = <виводиться результат обчислення>

Результат повинен містити три варіанти відповіді (по одному для кожної з гілок обчислювального процесу) і відповідні значення функції F = f (a, b, c, x), обчислені (для контролю) на калькуляторі.

2. Проаналізувати отримані вирази: визначити допустимі діапазони зміни вхідних величин, їх розмірність і тип.

3. Підготувати контрольні приклади (використовуючи, наприклад калькулятор), які повною мірою характеризують аналізовані вирази.

4. Розробити алгоритм обчислення і намалювати його графічну схему (блок-схему).

5. Відповідно до алгоритму набрати і відкомпілювати текст програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Аналіз вихідного виразу індивідуального завдання з обґрунтуванням контрольних прикладів, вибору типів даних і найбільш доцільною послідовністю