### Інструкції та блоки інструкцій

*Інструкція (statement)* – це ділянка коду, що описує дію, яку має виконати програма. Інструкції бувають прості (оголошення, розрахунок, присвоєння, виклик методу тощо) та складні (умова, цикл тощо). У кінці простої інструкції завжди є крапка з комою.

Розглянемо фрагмент коду:

int i = 10; i = i \* 2;

Він складається із двох простих інструкцій. У першій оголошено змінну цілого типу i та ініціалізовано її значенням 10. У другій поточне значення змінної i множиться на 2, а результат присвоюється тій же змінній i.

Загалом інструкції можна поділити на такі категорії:

► Інструкції оголошення. Описують оголошення типів або змінних.

► Вкладені інструкції. Виконують якісь дії або керують потоком виконання програми.

► Помічені інструкції. Керують переходами всередині потоку виконання.

*Блок інструкцій* – це послідовність інструкцій, оточена парою відповідних фігурних дужок (відкриваючою на початку і закриваючою в кінці блоку). В ряді випадків блок може бути порожнім (не містити інструкцій).

Синтаксично блок обчислюється як єдина вкладена інструкція.

На основі попереднього фрагменту коду можна створити такий блок:

{ int i = 10; i = i \* 2; }

Блок можна використовувати всюди, де синтаксис мови передбачає інструкцію, але дія, яку має виконати програма, містить кілька інструкцій. На відміну від простих інструкцій, блок не завершують символом крапки з комою.

Деякі програмні конструкції потребують блоку (наприклад, метод). У цьому випадку, замість блоку не можна підставити окрему інструкцію.

{

int i = 10;

i = i \* 2;

}

Кінець простої інструкції містить ";"

Кінець

блоку без символа ";"

Програма може містити також *порожні інструкції*, які містять тільки крапку з комою. Такі інструкції можна використовувати в коді у тих місцях, де синтаксис мови передбачає вбудовану інструкцію, але логіка алгоритму не потребує ніяких дій. Наприклад:

if( x < y ) // Перевіряємо умову { // Якщо умова виконується:

; // Порожня інструкція: нічого не робимо } else { // Якщо умова не виконується:

c = a + b; // Проста інструкція: обчислюємо вираз }

Інструкція може містити вираз, який розраховує певне значення. У програмі це значення вкладається у те місце, де розташований вираз. Тому говорять, що вираз *повертає* (у програму) розраховане значення певного типу. Останній рядок попереднього фрагменту коду обчислює суму значень змінних a та b, повертає результат (суму) і записує його у ділянку пам'яті, представлену змінною c.

### Керування потоком виконання

Сукупність інструкцій, які виконуються під час роботи програми, називають її *потоком виконання*. Зазвичай інструкції виконуються лінійно, послідовно одна за другою. Але виключно лінійне виконання інструкцій не може забезпечити потреб програмування. Потрібні засоби, які при потребі дозволять перенаправити потік виконання в іншу ділянку коду. Інструкції, які дозволяють це робити, називають інструкціями для керування потоком виконання.

Складні інструкції, які складаються з інших інструкцій, називають ще програмними *конструкціями*. C# забезпечує кілька конструкцій для керування потоком виконання, характерні і для інших сучасних мов програмування. До них належать:

► Конструкції умовного виконання. Вони виконують або пропускають секцію коду залежно від заданої умови. Є такі інструкції умовного виконання:

* if
* if … else
* switch

► Циклічні конструкції. Вони забезпечують багатократне виконання однієї і тієї ж ділянки коду. До них належать інструкції:

* while
* do
* for
* foreach

► Інструкції переходу. Вони дозволяють перейти від однієї секції коду до іншої. Це виконують такі інструкції:

* break
* continue
* return
* goto
* throw

Конструкції для умовного виконання та циклів містять вираз з умовою, яким задають, де саме продовжуватиметься виконання програми. Цей вираз повинен повертати результат типу bool. Вираз для умови може бути складний. Його можна формувати, поєднуючи прості умови за допомогою логічних операторів &&, || та !.

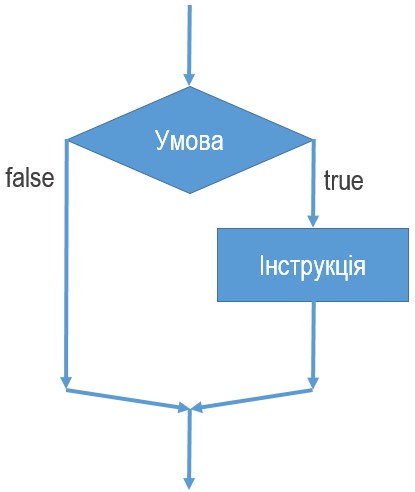
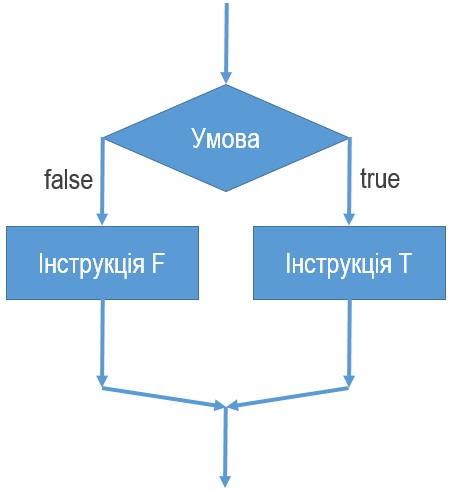
#### Інструкція if

Інструкція if реалізує виконання за умовою (рис. 2.8, а). Синтаксис інструкції такий:

if (TestExpression) Statement

Вираз TestExpression задає умову, від якої залежить перемикання на одну з двох віток потоку виконання. Вираз повинен повертати результат типу bool. Якщо TestExpression дорівнює true, то виконується інструкція Statement. Якщо умова обчислюється як false, то інструкція Statement пропускається. На місці інструкції Statement можна розмістити блок інструкцій, якщо логіка алгоритму передбачає більше однієї операції.

Лістинг 2.3. містить кілька прикладів використання інструкції if. Перший фрагмент коду (рядки 4…5) стосується випадку, коли при виконанні умови потрібно виконати тільки одну інструкцію. Якщо задовольняється умова (x<=10), то виконується інструкція в рядку 5, а після неї – наступні інструкції. Коли умова не виконується, інструкція в рядку 5 пропускається, і виконання продовжується з першої після рядка 5 інструкції (у даному випадку – 10).

а б

Рис. 2.8. Схема виконання інструкцій if (а) та if…else (б)

Другий фрагмент (рядки 10…14) ілюструє ситуацію, коли при виконанні умови, заданої в конструкції if, потрібно виконати кілька інструкцій. Якщо задовольняється умова (x>100), то виконується блок інструкцій (рядки 11…14); в іншому випадку блок пропускається, і виконання продовжується з першої після закінчення блоку інструкції.

|  |  |
| --- | --- |
| Лістинг 2.3. Приклади використання інструкції if | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // Приклад з однією інструкцією. Якщо при виконанні умови  // потрібно виконати тільки одну інструкцію, її не оточують // фігурними дужками if (x <= 10) y = x + 50; // Єдина інструкція  // Приклад з кількома інструкціями. Якщо при виконанні умови  // слід виконати більше однієї інструкції, то формують блок, // оточуючи його фігурними дужками if (x > 100) { x = x – 5; // Кілька інструкцій оточують дужками, y = x + z; // формуючи з них блок інструкцій  } |

#### Інструкція if…else

Інструкція if…else розділяє потік виконання на дві вітки, і перемикання на одну з них керується виконанням умови (рис. 2.8, б). Синтаксис конструкції такий:

if (TestExpression) StatementT else StatementF

Якщо задовольняється умова, задана виразом TestExpression (тобто TestExpression повертає true), то виконується інструкція StatementT. В іншому випадку виконується інструкція StatementF. Якщо алгоритм передбачає виконання більше однієї інструкції при виконанні (не виконанні) умови, то замість одиничної інструкції використовують блок інструкцій.

Приклади використання інструкції if…else приведено в лістингу 2.4.

Конструкція на основі if…else розділяє потік виконання на дві вітки. Якщо потрібно створити більше віток, можна використати вкладені інструкції (if…else) (рядки 11…16).

|  |  |
| --- | --- |
| Лістинг 2.4. Приклади використання інструкції if…else | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | // Конструкція if...else if (x < 10 ) y = x – 1; // Єдина інструкція else { // Блок інструкцій x = x – 5; y = x + z; }  // Вкладені конструкції if...else if (TestExpression1) Statement1  else if (TestExpression2) Statement2 else  Statement3 |

#### Інструкція switch

Інструкція switch забезпечує множинне розгалуження потоку виконання (рис. 2.9). Вона містить різні секції коду, кожна з яких починається відповідною міткою з ключовим словом case. Будь-яка секція коду повинна закінчуватися інструкцією break, або іншою інструкцією переходу (return, continue, goto, чи throw). Зі всіх інструкцій переходу найчастіше використовують break. Вона припиняє виконання секції і переходить до кінця конструкції switch. Кожна мітка у конструкції switch має таку структуру:

case ConstantN

:

Ключове слово

Двокрапка: кінець мітки

Константа

Конструкція switch виконується так:

Спочатку обчислюється вираз TestExpression, який вказано в інструкції switch.

Якщо значення виразу TestExpression дорівнює значенню константи ConstantX однієї з міток, то виконуються інструкції з секції, відміченої цією міткою. Коли ні одна з міток не містить константи, яка дорівнює значенню виразу TestExpression, виконуються інструкції з секції default.

При необхідності секцію default можна не включати у конструкцію switch. Але, якщо вона є, то має закінчуватися, як і інші секції, інструкцією break, або іншою інструкцією переходу (return, continue, goto, чи throw)[[1]](#footnote-1).

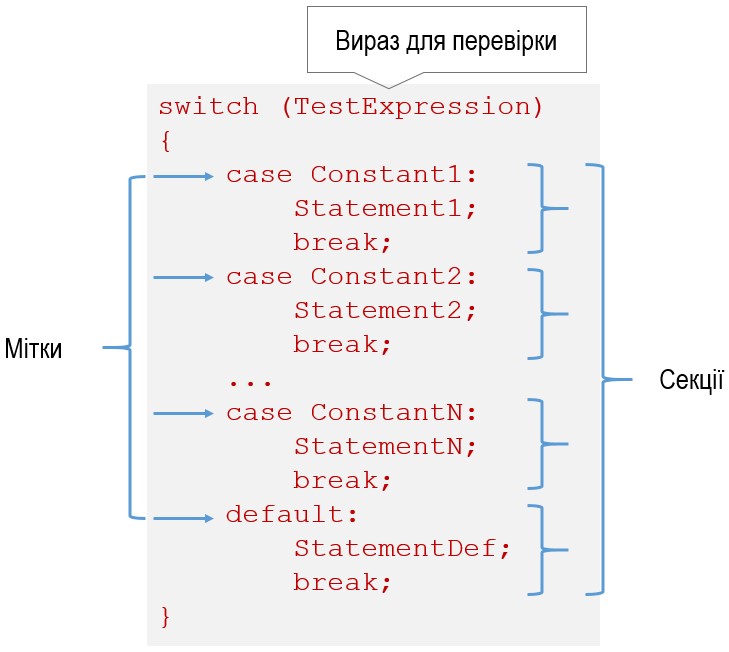
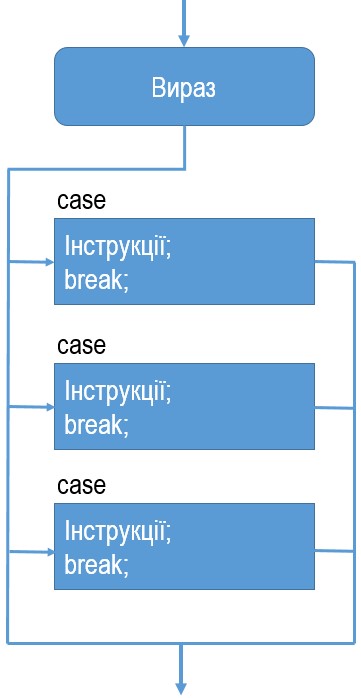


Рис. 2.9. Структура інструкції switch

#### Цикл while

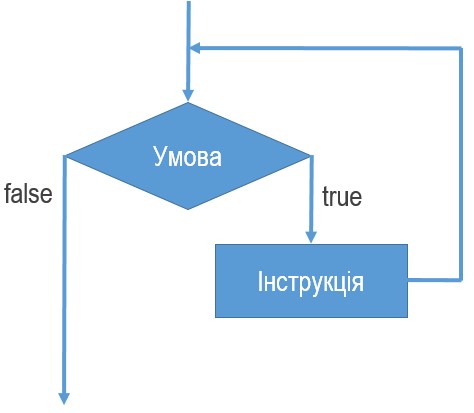
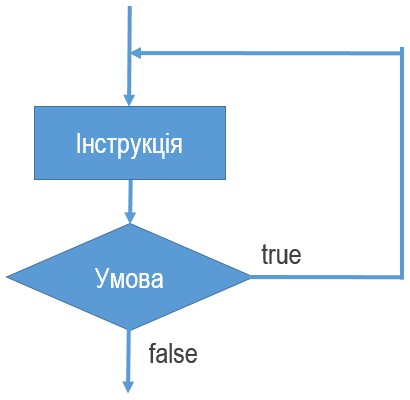
Конструкція while формує цикл, у якому умова, що визначає його виконання (чи невиконання), перевіряється на початку циклу (рис. 2.10, а).

Циклічну конструкцію, в якій умову виконання циклу перевіряють на початку циклу, називають *циклом з передумовою*.

Синтаксис використання інструкції while такий:

while (TestExpression) Statement

Вираз для перевірки умови виконання циклу TestExpression повинен повертати результат типу bool. При досягненні у потоці виконання інструкції while спочатку обчислюється значення виразу TestExpression. Якщо результат TestExpression дорівнює true, виконується інструкція Statement, що формує тіло циклу. Після цього знову розраховується значення виразу TestExpression, і все повторюється спочатку. Якщо вираз TestExpression повертає значення false, виконання циклу припиняється, і виконання програми переміщується до першої інструкції після конструкції циклу.

а б

Рис. 2.10. Структура циклів while (а) та do (б)

Коли в циклі слід виконувати більше однієї інструкції (як правило, так і

є), на місці інструкції Statement слід сформувати блок інструкцій.

Зауважимо, що коли при першому входженні у цикл while умова циклу не задовольняється, то тіло циклу не буде виконане жодного разу.

Можлива і протилежна ситуація: якщо умова циклу виконується завжди, то потік виконання, зайшовши у цикл, ніколи з нього не вийде, постійно виконуючи його інструкції. Такої ситуації слід уникати, оскільки вона приведе до "зависання" програми. Тому при проектуванні циклічних конструкцій слід ретельно продумати умови припинення роботи циклу.

У лістингу 2.5 приведено приклад використання циклу while. В рядку 1 оголошено дійсні змінні x, xMax та dX, які відразу ініціалізовано певними значеннями. Умова виконання циклу (x < xMax) забезпечує постійне циклічне виконання тіла циклу (блок у рядках 4…7), доки значення змінної x менше за значення xMax. У тілі циклу спочатку виводимо на екран поточне значення x (рядок 5), потім прирощуємо x на значення dX (рядок 6)[[2]](#footnote-2). Результат виконання програми з лістингу 2.5 приведено на рис. 2.11.

|  |  |
| --- | --- |
| Лістинг 2.5. Приклад використання циклу while | |
| 1  2  3  4  5  6  7 | double x = 0, xMax = 5.5, dX = 0.99; while (x < xMax) { Console.WriteLine("x = {0}", x); x += dX;  } |

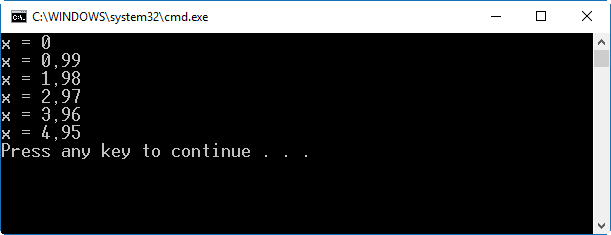


Рис. 2.11. Результат роботи циклу while з лістингу 2.5

#### Цикл do

Цикл do, на відміну від while, перевіряє умову виконання циклу в кінці (рис. 2.10, б). Циклічну конструкцію, у якій умову перевіряють в кінці циклу, називають *циклом з післяумовою*. Синтаксис використання циклу do такий:

do Statement

while (TestExpression); // Кінець циклу

При відпрацюванні такого циклу спочатку виконується інструкція (або інструкції) Statement циклу, а потім обчислюється умова циклу – вираз TestExpressison. Якщо вираз TestExpressison повертає значення true, інструкція Statement виконується ще раз. Цикл повторюється, доки TestExpressison дорівнює true. Коли вираз із умови циклу поверне результат false, виконання інструкцій Statement припиняється, і виконуються інструкції, розташовані після циклу.

Особливістю циклу do є те, що тіло циклу виконується хоча б один раз в будь-якому випадку, навіть якщо до входження у конструкцію циклу умова циклу не задовольняється.

Приклад використання циклу do приведено в лістингу 2.6, а результат його виконання – на рис. 2.12.

|  |  |
| --- | --- |
| Лістинг 2.6. Приклад використання циклу do | |
| 1  2  3  4  5  6 | double x = 1, xMax = 5.5, dX = 1.5; do { Console.WriteLine("x = {0}", x); x += dX;  } while (x <= xMax); // <-- Крапка з комою потрібна |

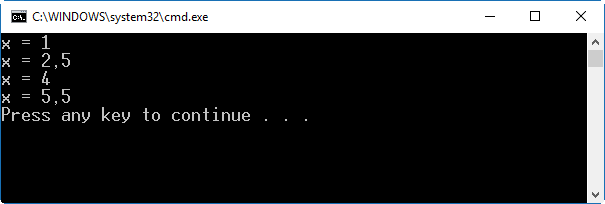


Рис. 2.12. Результат виконання циклу do з лістингу 2.6

#### Цикл for

Конструкція на основі інструкції for виконує цикл з передумовою, але додатково містить вирази для ініціалізації та ітерації. Схему роботи такого циклу зображено на рис. 2.13. Його синтаксис:

Вирази розділені символом

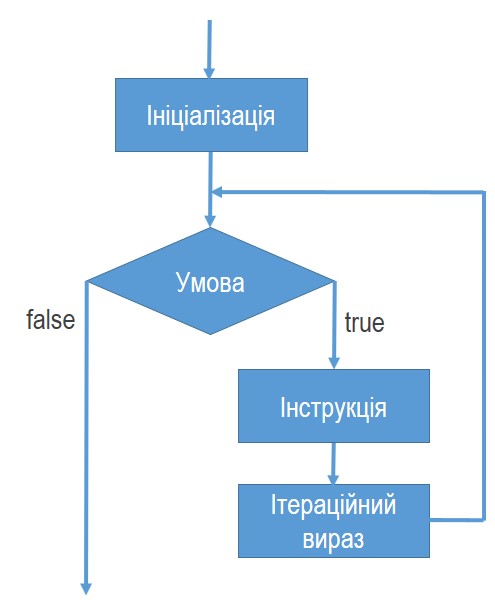
"

**;**

"

for (Initializer ; TestExpression ; IterationExpression) Statement

Перед виконанням циклу відбувається ініціалізація циклу – один раз виконується вираз Initializer. Далі обчислюється умова циклу TestExpression. Якщо вираз TestExpression повернув true, виконується інструкція у тілі циклу. Після цього виконується ітераційний вираз IterationExpression. Далі виконання знову переходить до початку циклу, і повторно обчислюється умова TestExpression. Доки вираз умови циклу повертає true, виконується інструкція (інструкції) у тілі циклу, після чого завжди виконується ітераційний вираз. Виконання циклу припиняється, коли TestExpression повертає false.

 Рис. 2.13. Схема циклу for

Вирази Initializer, TestExpression та IterationExpression необов'язкові. Їх позиції можна залишити порожніми, але розділювачі (символи ";") мають бути обов'язково. Якщо вираз TestExpression не вказати, то вважатиметься, що тестовий вираз завжди повертає true, – тоді слід передбачити інший спосіб припинення циклу.

Приклад використання циклу for приведено у лістингу 2.7, а результат його виконання – на рис. 2.14.

|  |  |
| --- | --- |
| Лістинг 2.7. Приклад використання циклу for | |
| 1  2  3  4 | for (int i = 1; i <= 5; i++) {  Console.WriteLine("i = {0}", i);  } |

У цьому коді на початку циклу змінна i ініціалізується значенням 1. Цикл виконується, доки значення i <= 5. В кінці кожного проходження тіла циклу за допомогою ітераційного виразу (i++) значення змінної циклу i збільшується на 1.

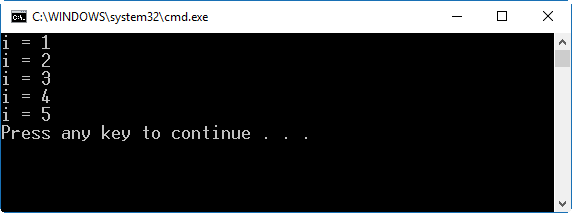


Рис. 2.14. Результат виконання циклу for з лістингу 2.7

#### Інструкції переходу

Коли потік виконання програми досягає інструкції переходу, виконання програми безумовно (без перевірки будь-якої умови) переноситься до іншої ділянки програми.

У С# є такі інструкції переходу:

► break;

► continue;

► return;

► goto;

► throw;

#### Інструкція break

Інструкцію break можна використовувати у конструкціях while, do, for, foreach та switch для переривання виконання конструкції. Якщо програма містить вкладені конструкції, то break припиняє виконання найглибшої вкладеної конструкції. Таким чином, якщо є два цикли, один вкладений в інший, і break викликається у тілі вкладеного циклу, то відбувається вихід із вкладеного циклу, але не із зовнішнього.

У лістингу 2.8 приведено приклад використання break для припинення виконання нескінченного циклу.

|  |  |
| --- | --- |
| Лістинг 2.8. Використання інструкції break для переривання циклу | |
| 1  2  3  4  5  6 | int x = 0; while (true) { x++; if (x == 3) break;  } |

Цикл (рядки 2…6) містить умову, яка завжди повертає значення true (вона задана в рядку 2). Без додаткових заходів такий цикл виконуватиметься "вічно", що спричинить "зависання" програми. Щоб не допустити цього, в циклі перевіряємо значення змінної x (яка, починаючи з нуля, при кожному проходженні циклу прирощує своє значення на 1). Коли x досягає значення 3, виконання циклу припиниться, оскільки виконається інструкція break.

#### Інструкція continue

Інструкцію continue використовують у циклах while, do, for та foreach. Вона спричиняє перехід до початку (найглибшого вкладеного) циклу.

Розглянемо приклад її використання у лістингу 2.9. Цикл, як задано у конструкції for, виконується 7 разів (рядок 1). Але для перших трьох разів, оскільки x буде меншим або дорівнюватиме 3, виконуватиметься умова if (рядок 3), після чого в результаті спрацювання інструкції continue виконання переходитиме на початок циклу. Тому інструкції з тіла циклу, розташовані після рядка 4, не виконуватимуться. Таким чином, вивід на екран значення x (в рядку 7) буде тільки для тих x, які більші від 3. Результат виконання лістингу 2.9 приведено на рис. 2.15.

|  |  |
| --- | --- |
| Лістинг 2.9. Використання інструкції continue у циклі for | |
| 1  2  3  4  5  6  7  8 | for (int x = 1; x <= 7; x++) // Цикл виконується 7 разів { if (x <= 3) // Перші 3 рази continue; // відразу переходимо до початку циклу  // Цей рядок виконуватиметься, якщо x >= 3  Console.WriteLine("x = {0}", x);  } |

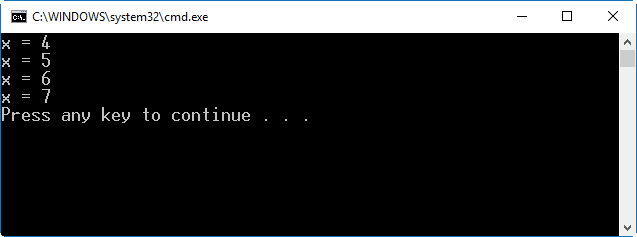


Рис. 2.15. Результат виконання коду з лістингу 2.9

|  |  |
| --- | --- |
| Лістинг 2.10. Використання інструкції continue у циклі while | |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int x = 1; while (x <= 7) { if (x <= 3) { x++; continue; // Перехід на початок циклу }  // Цей рядок виконуватиметься, якщо x >= 3 Console.WriteLine("x = {0}", x); x++;  } |

У лістингу 2.10 приведено приклад використання інструкції continue у циклі while. Результат його роботи буде такий самий, як і в попередньому випадку (рис. 2.15).

#### Мітки

*Мітка* – це інструкція, яка містить ідентифікатор з двокрапкою. Її використовують для безумовного переходу до того місця коду, яке вона позначає. Синтаксис використання мітки такий:

LabelName: Statement

де LabelName – назва мітки. Помічена інструкція Statement виконується так, ніби мітки не існує. Мітки оголошують лише всередині блоків коду. Область видимості мітки охоплює блок коду, в якому вона оголошена, та всі вкладені у нього (внутрішні) блоки коду.

Для переходу з будь-якого місця коду (в області видимості мітки) до виконання поміченої інструкції використовують інструкцію goto: goto LabelName;

Після переходу на помічену інструкцію вона виконується, і далі виконання програми продовжується з цього місця коду.

Зауважимо, що інструкція goto часто заплутує код, оскільки сприяє поганій структуризації коду і як наслідок – погіршує його розуміння. Вважається хорошим тоном програмування по можливості не використовувати інструкцію goto.

Лістинг 2.11 демонструє дію мітки. У рядку 3 програма входить у "нескінченний" цикл (умова циклу завжди true). У циклі виводиться на екран поточне значення змінної i – ітератора циклу. А в рядку 7 виконується перевірка, чи значення i дорівнює 6. Якщо так, то за допомогою інструкції goto здійснюється перехід до інструкції в рядку 14, яка помічена міткою в рядку 13. Результат роботи коду з лістингу 2.11 показано на рис. 2.16.

|  |  |
| --- | --- |
| Лістинг 2.11. Використання мітки | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int i = 0; Console.WriteLine("Починаємо нескiнченний цикл"); while (true) { i++; Console.WriteLine("i = {0}", i); if (i == 6) { Console.WriteLine("Пора закiнчувати..."); goto TheEnd; }  } TheEnd:  Console.WriteLine("Кiнець програми..."); |

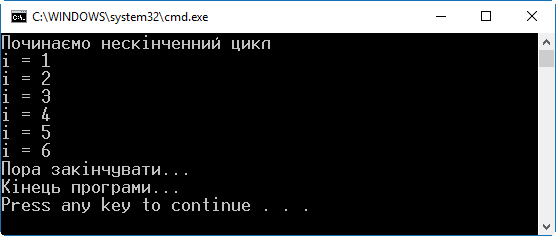


Рис. 2.16. Результат виконання коду з лістингу 2.11

#### Інші інструкції

Мова C# містить також інші інструкції, які керують потоком виконання програми. Серед них – foreach, try, throw, finally, return. Вони тісно пов'язані з різними особливостями програмування, тому розглянемо їх під час обговорення відповідних питань у наступних розділах.

1. Детально про інструкції переходу йтиме мова далі в цьому розділі.

   [↑](#footnote-ref-1)
2. 12 Оператор += описано у таблиці 2.17. [↑](#footnote-ref-2)