Інструкції

- Види інструкцій C#
- Загальні зауваження щодо інструкцій
- Умовна інструкція if
- Перемикач switch
- Покроковий цикл for
- Зауваження щодо for
- Цикл foreach
- Цикл з передумовою while
- Зауваження щодо while
- Цикл з післяумовою do-while
- Зауваження щодо do-while
- Прийоми використання циклів
- Введення\виведення з консолі

Види інструкцій C# для задання алгоритмів

Лінійні алгоритми

- Визначення (оголошення)
- **expr;** інструкція-вираз
- **{ }** складена інструкція
- **;** пуста інструкція

Галужені алгоритми

- **if** умовна інструкція
- **switch** перемикач
- **continue** інструкція продовження
- **break** інструкція виходу
- **return** інструкція повернення

Циклічні алгоритми

- **for** покроковий цикл
- **foreach** цикл по елементах переліку(масиву/колекції)
- **while** цикл з передумовою
- **do-while** цикл з післяумовою

Загальні зауваження щодо інструкцій

- Усі інструкції, крім блоку, завершуються символом **;**
- Обчислене значення **expr** може ігноруватится
- Інструкція-оголошення локалізує точку введення об'єкта в область видимості

Умовна інструкція if

```
if( expr ) instr1 ;

if( expr ) instr1 else instr2 ;
```

Алгоритм :

1. обчислюється **expr**
   і результат трактується (без приведення) як **bool**
2. якщо отриманий результат **true** , то виконується **instr1** ,
   інакше виконується **instr2** (якщо вона наявна)

- Якщо наявні **instr1 і instr2,** то виконається лише одна з них
- Якщо варіанти **instr1 і instr2** задають кілька дій, то їх оформляють у блок
- У повному **if instr1** завершується **;** (якщо не є блоком)
- Об’єкти, оголошені у **expr** , мають область дії до кінця **instr2**
- Конструкція **else** асоціюється з найближчою попередньою конструкцією **if**, яка явно не пов’язана з іншою **else**

Перемикач `switch`

```
switch( expr ){
    case const-expr1 : instr-list1
                       break;
    case const-expr2 : instr-list2
                       break;
    case const-exprk : instr-listk
                       break;
    default:  instr-list
}
```

Алгоритм перемикача

Алгоритм :

1. обчислення **expr**
2. співставлення отриманого результат зі значенням константних виразів у заданому порядку
3. при першому співпадінні значень виконання відповідних інструкцій,

   інакше виконання інструкцій *за замовчуванням* (якщо вони наявні)

- Якщо **break** (або **return**) відсутні серед інструкцій виконуваного варіанту**,** то після його завершення почнеться виконання інструкцій наступного варіанту
- Результат **expr** має бути інтегрального типу
- Має існувати неявне перетворення типу кожного константного виразу до типу **expr**
- Порядок **case**-варіантів є суттєвим
- Варіант **default** опційний(може бути відсутній)
- Об’єкти, оголошені у **expr** , мають область дії до кінця перемикача

Покроковий цикл for

```
for( init-expr ; cond-expr ; loop-expr ) instr ;
```

Алгоритм :

1. Обчислення **init-expr**
2. Обчислення **cond-expr**
3. Якщо результат попереднього обчислення є **false**, то перехід на наступну після **for** інструкцію (завершення циклу)
4. інакше :

    4.1. виконання **instr**

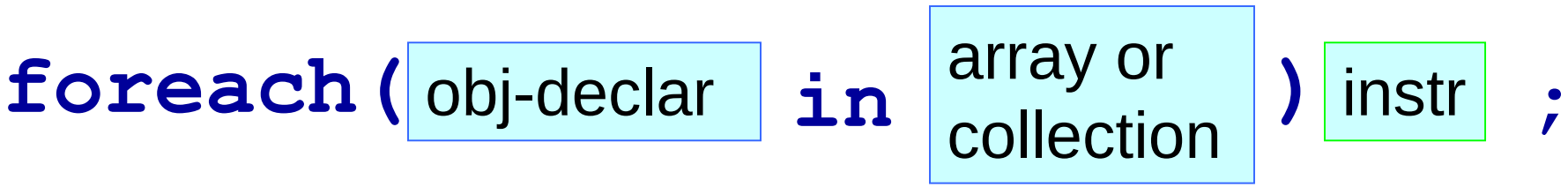
    4.2. обчислення **loop-expr**

    4.3. перехід на **#2**

Зауваження щодо for

- **init-expr о**бчислюється лише 1 раз
- результат **init-expr** використовують (як правило) для ініціалізації *змінної циклу*
- значення **true cond-expr** є умовою виконання тіла циклу
- відсутність **cond-expr** еквівалентна **true**
- цикл з передумовою, тому **instr** може не виконуватися жодного разу
- **loop-expr** використовують для модифікації змінної циклу
- **for(;;){ ... } -** " вічний " цикл
- **break** або **return** під час виконання **instr** завершують виконання циклу
- **continue** під час виконання **instr** завершує поточну ітерацію і передає керування на #4.2.

Цикл foreach



Алгоритм :

1. Якщо в переліку (масив або колекція) немає доступних елементів, перехід на наступну після **foreach** інструкцію (завершення циклу)
2. інакше :

    2.1. посилання налаштовується на перший доступний елемент

    2.2. виконання **instr**

    2.3. перехід на **#1**

Цикл з передумовою while

```
while( expr ) instr ;
```

Алгоритм :

1. Обчислення **expr**
2. Якщо результат попереднього обчислення є **false**, то перехід на наступну після **while** інструкцію (завершення циклу)
3. інакше :

   3.1. виконання **instr**

   3.3. перехід на **#1**

Зауваження щодо while

- **expr о**бчислюється на початку кожної ітерації
- значення **expr true** є умовою виконання тіла циклу
- наявність **expr** обов'язкова
- цикл з передумовою, тому **instr** може не виконуватися жодного разу
- під час виконання **instr** (або **о**бчислення **expr)** компоненти **expr** мають модифікуватися – інакше " вічний " цикл
- **break** або **return** під час виконання **instr** завершують виконання циклу
- **continue** під час виконання **instr** завершує поточну ітерацію і передає керування на #1.

Цикл з післяумовою do-while

```
do instr ; while( expr );
```

Алгоритм :

1. Виконання **instr**
2. Обчислення **expr**
3. Якщо результат попереднього обчислення є **false**, то перехід на наступну після **do-while** інструкцію (завершення циклу)
4. інакше перехід на **#1**

Зауваження щодо do-while

- **expr о**бчислюється після виконання **instr**
- значення **expr** **true** є умовою виконання тіла циклу
- наявність **expr** обов'язкова
- цикл з післяумовою, тому **instr** буде виконуватися принаймні 1 раз
- під час виконання **instr** (або **о**бчислення **expr)** компоненти **expr** мають модифікуватися – інакше " вічний " цикл
- **break** або **return** під час виконання **instr** завершують виконання циклу
- **continue** під час виконання **instr** завершує поточну ітерацію і передає керування на #2.

Прийоми використання циклів

1. **Вибір виду інструкції циклу**
   - **for** - відома наперед к-сть ітерацій
   - **while** - к-сть ітерацій наперед невідома, їх може бути 0
   - **do_while** - відбудеться принаймні 1 ітерація, але їх к-сть невідома
   - **foreach** – ітерація по усіх елементах колекції
2. **Очевидний спосіб керування ітераціями**
   - **break** - достроковий вихід з циклу
   - **continue** - завершення поточної ітерації
3. **Код ініціалізації розміщувати безпосередньо**
   - перед інструкцією **while** або **do_while**
   - у заголовку **for**
4. Службові інструкції керування циклом групувати в одному місці
5. Вирази в умові повторення ітерацій мають бути простими
6. Керуючі змінні циклу не використовувати не за призначенням
7. Глибина вкладення циклів <=3

Output to the Console

```
public static void Write( char value );//overloaded for native types

public static void Write( string format, Object arg );

public static void WriteLine( char value ); // Writes the specified Unicode character,
                                            // followed by the current line terminator, value to the standard output stream.
```

```
Console.Write(intVal);          // overloaded for all primitive types
Console.WriteLine(intVal);      // for objects ToString() is called automatically
```

```
Console.Write("Hello {0}", name);  // placeholder
Console.WriteLine("{0} = {1}", x, y);
Console.WriteLine($"{x} = {y}");
```

Placeholder syntax

"{" n ["," width] [":" format [precision]] "}"

n	argument number (starting at 0)
width	field width (exceeded if too small) positive = right-aligned, negative = left-aligned
format	formatting code (e.g. d, f, e, x, ...)
precision	number of fractional digits (sometimes number of digits)

Example: **{0, 10:f2}**

Formatting Codes for Numbers

d, D	decimal format (integer number with leading zeroes) precision = number of digits	-xxxxx
f, F	fixed-point format precision = number of fractional digits (default = 2)	-xxxxx.xx
n, N	number format (with separator for thousands) precision = number of fractional digits (default = 2)	-xx,xxx.xx
e, E	floating-point format (case is significant) precision = number of fractional digits	-x.xxxE+xxx
c, C	currency format precision = number of fractional digits (default = 2) negative values are enclosed in brackets	$xx,xxx.xx ($xx,xxx.xx)
x, X	hexadecimal format (case is significant) precision = number of hex digits (maybe leading 0)	xxx
g, G	general (most compact format for the given value; default)	

Examples

```
int x = 17;

Console.WriteLine("{0}", x);       17
Console.WriteLine("{0,5}", x);        17

Console.WriteLine("{0:d}", x);     17
Console.WriteLine("{0,5:d3}", x);    017

Console.WriteLine("{0:f}", x);     17.00
Console.WriteLine("{0:f1}", x);    17.0

Console.WriteLine("{0:E}", x);     1.700000E+001
Console.WriteLine("{0:e1}", x);    1.7e+001

Console.WriteLine("{0:x}", x);     11
Console.WriteLine("{0:x4}", x);    0011
```

String Formatting

- With `ToString` for numeric types (`int, long, short, ...`):

```
string s;
int i = 12;
s = i.ToString();      // "12"
s = i.ToString("x4"); // "000c"
s = i.ToString("f");  // "12.00"
```

- With `String.Format` for arbitrary types

```
s = String.Format("{0} = {1,6:x4}", name, i);    // "val =   000c"
```

- Culture-specific formatting

```
s = i.ToString("c");                    // "$12.00"
s = i.ToString("c", new CultureInfo("en-GB"));   // "£12.00"
s = i.ToString("c", new CultureInfo("de-AT"));   // "€12.00"
```

Keyboard Input

- **`Console.Read();`**
  - returns the next character
  - waits until the user pressed the return key

e.g. input: "abc" + return key
Read returns: 'a', 'b', 'c', '\r', '\n'.

- **`Console.ReadLine();`**
  - returns the next line (after Ctrl-Z+CR+LF it returns null)
  - waits until the user pressed the return key
  - returns the line without CR, LF