Тема 9. Використання операторів циклу в програмуванні

1. **Цикли з перед- та післяумовою.**
2. **Цикли з параметром.**
3. **Перелічувальний оператор циклу.**
4. **Вкладені цикли.**

1. Оператори циклу дають змогу виконати вкладені оператори декілька разів, при цьому програміст сам повинен подбати, щоб рано чи пізно відбувся вихід циклу, інакше виникне зациклення і програма зависне.

Цикли з передумовою в C# записуються згідно синтаксису:

```
while(<умова>)
        <оператор>;
```

При виконанні цього оператора перевіряється істинність умови. Якщо умова істинна, то виконується вкладений оператор і знову відбувається перевірка істинності умови і так далі. Цикл завершує виконання тоді, коли умова виявляється хибною.

Синтаксис циклу з післяумовою:

```
DO
{<оператори>;
}while(<умова>);
```

При виконанні цього циклу спочатку виконуються вкладені оператори, після чого перевіряється істинність умови. Якщо умова істинна, то оператори виконуються ще раз і т. д. вкладені оператори виконуються до тих пір, поки умова залишається істинною.

На відміну від циклу з передумовою в циклі з післяумовою вкладені оператори виконуються хоча б один раз.

Приклад:

Протабулювати значення функції $y = \frac{x + \cos 2x}{3x}$ на проміжку $2.3 \le x \le 5.4$ кроком $\Delta x = 0.8$. Результат вивести у вікні консольного додатку.

```
Double x, y, xp, xk, dx;
xp = 2.3;
xk = 5.4;
dx = 0.8;
Console.Write("Введіть x початкове ");
xp = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть x кінцеве ");
xk = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть крок ");
dx = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("x   y");
x = xp;
while (x<=xk)
{ y = (x+Math.Cos(2*x))/(3*x);
  Console.WriteLine("{0} {1}", x, y);
  x += dx;  (або x=x+dx;)
}
Console.ReadKey();
```

Якщо при виконанні циклу необхідно його перервати, то використовується оператор `break`. Коли ж необхідно перейти до наступної ітерації циклу, тобто не виконувати наступні оператори, а повернутися до перевірки виконання умови, то застосовують оператор `continue`.

Приклад:

Обчислити значення функції $y = \frac{x + \cos 2x}{3x}$, починаючи з початкової точки $1.2 \le x$ з заданим кроком $\Delta x = 0.2$. у вказаній кількості точок n = 9.

```
Int n, i;
Double x, y, xp, dx;
xp = 1.2;
dx = 0.2;
n = 9;
Console.Write("Введіть x початкове ");
xp = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть крок ");
dx = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть кількість точок");
n = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("x   y");
x = xp;
i = 0;
while (i<=n)
{ y = (x+Math.Cos(2*x))/(3*x);
  Console.WriteLine("{0} {1}", x, y);
  x += dx;
}
Console.ReadKey();
```

Приклад:

При заданому x обчислити наближену суму $S = e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1} + \frac{x^2}{1*2} + \frac{x^3}{1*2*3} + \frac{x^4}{1*2*3*4}$, припиняючи обчислення, коли черговий член менше 0.001.

```
Dodatok = 1;
Nomer = 1;
S = 0;
while(Math.Abs(Dodatok)>0.001)
{ S+=Dodatok;
  Dodatok*=x/Nomer;
  Nomer++;
}
```

2. В C# як і в класичному C++ цикл з параметром записується згідно синтаксису:

```
for (<початкові значення параметрів>;
        <умова виконання тіла циклу>;
        <зміни параметрів>)
        <тіло циклу>;   //якщо декілька дій.
```

На початку виконання оператора в параметри заносяться початкові значення, після цього перевіряється істинність умови. Якщо умова істинна, то виконується тіло циклу і змінюються параметри у відповідності з 3-тім аргументом `for` і т. д. Цикл виконується до тих пір поки умова не стане хибною. В циклі може використовуватися декілька початкових значень і декілька змін параметрів, які між собою відмежовуються комами.

Приклад:

Обчислити значення функції у заданій кількості точок з вказаним кроком.

$y = \frac{\sin^2 x}{(x+1)^3}$ $x \geq 0.5$ $\Delta x = 0.1$ $n = 9$

Параметри обчислень і результати ввести/вивести за допомогою діалогових вікон.

x = xp + i*step i – вказує на кількість виведених точок

```
int n, i;
Double x, y, xp, step;
String S;
xp = 0.5;
```

```
   step = 0.1;
   n = 9;
Povtor:
   if (!InputDouble(ref xp, "Введіть початок проміжку"))
      return;
   if (!InputDouble(ref step, "Введіть крок"))
      return;
   if (!InputInt(ref n, "Введіть кіськість точок"))
      return;
   S = "x        y";
   for (i = 0; i < n; i++)
   {
     x = xp + i * step;
     y = Math.Pow(Math.Sin(x),2)/Math.Pow((x+1),3);
     S += Strings.Chr(13) + x.ToString() + "           " +
   y.ToString();
   }
   if (MessageBox.Show(S + Strings.Chr(13) + Strings.Chr(13) +
   "Бажаєте рахувати ще?", "Результати обчислень",
   MessageBoxButtons.YesNo) == DialogResult.Yes)
     goto Povtor;
```

Приклад:

Піднести число 3 до степеня 9, не використовуючи функцію піднесення до степеня.

$3^9 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$

```
   int osnova, stepin, dobutok;
   osnova = 3;
   stepin = 9;
   if (!InputInt(ref osnova, "Введіть крок"))
     return;
   if (!InputInt(ref stepin, "Введіть кіськість точок"))
     return;
   dobutok = 1;
   for (int i = 0; i< stepin; i ++ )
     dobutok*=osnova;
   MessageBox.Show("Якщо основа становить " +
   osnova.ToString() + " , степінь становить " +
   stepin.ToString() + "то добуток становить " +
   dobutok.ToString() + Strings.Chr(13) + Strings.Chr(13) +
   "", "Результати обчислень", MessageBoxButtons.OK);
```

Приклад:

Підрахувати суму натуральних чисел, що діляться на 3 і не перевищують n.

```
   int n, i;
   Double S;
   S = 0;
   if (!InputInt(ref n, "Введіть число не перевищення"))
     return;
   if (!InputInt(ref i, "Введіть число-дільник"))
     return;
   for (int i = 3; i< n; i += 3 )
     S+=i;
   MessageBox.Show("Якщо число не перевищення становить " +
   n.ToString() + " , число-дільник становить " + i.ToString()
   + "то сума натуральних чисел становить " + S.ToString() +
   Strings.Chr(13) + Strings.Chr(13) + "", "Результати
   обчислень", MessageBoxButtons.OK);
```

Приклад:

Обчислити середнє геометричне чисел не парних з проміжку [10;90].

$\sqrt[i]{a_1 \times a_2 \times \cdots \times a_i}$

```
int Dobutok=1, Count=0, i;
Double REZ;
for (i = 11; i<90; i +=2 )
{
  Dobutok *=i;//Обчислюємо добуток з черговим не парним числом
  Count++;//Збільшуємо кількість знайдених не парних чисел
}
REZ = Math.Pow(Dobutok,1/Count);
MessageBox.Show("Середнє геометричне чисел не парних з проміжку [10;90] становить " + REZ.ToString() + Strings.Chr(13) + Strings.Chr(13) + "", "Результати обчислень", MessageBoxButtons.OK);
```

Цикли та оператори розгалуження можуть вкладатися в один залежно від потреб алгоритму розгалуження задачі.

Приклад:

З клавіатури вводиться n стипендій (число n теж вводиться з клавіатури). Визначити мінімальну та максимальну стипендію. Масиви не використовувати.

```
int n = 10;//Значення кількості стипендій по замовчуванню
Double St = 700, MinSt = 0, MaxSt = 0;
int i;
if (!InputInt(ref n, "Введіть кількість стипендій"))
  return;
for (i = 1; i <= n; i++)
{
  if (!InputDouble(ref St, "Введіть " + i.ToString() + "-у стипендію"))
    return;
  if (i == 1)
  {
    MinSt = St;
    MaxSt = St;
  }
  else
  {
    if (St < MinSt)
      MinSt = St;
    if (St > MaxSt)
      MaxSt = St;
  }
}
MessageBox.Show("З кількості стипендій " + St.ToString() + " максимальна стипендія становить " + MaxSt.ToString() + ", а мінімальна стипендія становить " + MinSt.ToString() + Strings.Chr(13) + Strings.Chr(13) + "", "Результати обчислень", MessageBoxButtons.OK);
```

3. Крім циклів з перед- та післяумовою в програмуванні використовується перелічувальні цикли. Синтаксис перелічувального циклу має такий вигляд:

```
foreach (<тип даних><змінна циклу>in<сукупність>)
          <тіло циклу>
```

Цей цикл почергово заносить в значення змінної елементи з сукупності. Ми використаємо цей цикл при обробці елементів масиву.

4. Вкладені цикли

Цикли можна застосовувати і в середині інших циклів. Такі цикли називаються **вкладеними**. Вкладений цикл має використовувати інші параметри стосовно основного циклу. У випадку використання вкладених циклів для кожного значення основного циклу щоразу повторюється вкладений цикл.

Приклад:

Число досконале, якщо воно рівне сумі своїх дільників. Серед чисел від 1 до n знайти всі досконалі.

В змінну REZ(текстова) занесемо всі знайдені досконалі числа.

```
REZ = "";//поки досконалі числа не знайдені
for (chuslo=2; chuslo<=n; chuslo++)
{
 sumaD=1;//початкове значення суми дільників
 for (dilnuk=2; dilnuk<=chuslo/2; dilnuk++)
   if ((chuslo%dilnuk)==0)
     sumaD+=dilnuk;
   if (chuslo==sumaD)
     REZ+=chuslo.ToString() + Strings.Chr(13);
}
if (REZ=="")
   REZ="Досконалі числа відсутні";
```

Приклад:

Дано натуральне число N. Вияснити чи можна надати його у вигляді сум кубів двох натуральних чисел: N=x^3+y^3. Якщо це можливо, то вказати всі пари таких чисел x та y.

```
<вводимо N>
Count=0;
S:=””;
M:=Math.Exp(1.0/3*Math.Log(N));
FOR (x=0, x<=M; x++)
 FOR (y=0, y<=Mж н++)
  IF (N=x*x*x+y*y*y)
   {count++;
    S+=x.ToString() +”  ” + y.ToString() + ”\n”; }
If (count==0)
 MessageBox.Show(”Пари чисел відсутні”)
Else
 MessageBox.Show(”Знайдено ”+count.ToString()+” пар: \n” +S);
```

Підготовка до МКР2:

Приклад 1:
Знайти суму цілочисельних значень з проміжку [-19.1; 8.75]

```
Suma = 0;
for (i = -19; i <= 8; i++)
  Suma +=i;
MessageBox.Show…
```

Приклад 2:
Обчислити добуток $sin1^0 * sin2^0 * \cdots * sin45^0$.

```
Dobutok = 1;
for (i = 1; i <= 45; i++)
  Dobutok *= Math.Sin(i*180/3.14);
MessageBox.Show…
```

Приклад 3:
Обчислити суму квадратів чисел -8, -6, … , 10, 12.

```
Suma = 0;
for (i = -8; i <= 12; I += 2)
  Suma +=i*i;
MessageBox.Show…
```

Приклад 4:
Обчислити суму членів ряду $\frac{2}{1!}+\frac{3}{2!}+\frac{4}{3!}+\cdots+\frac{n+1}{n!}$.

```
Suma = 0;
Factorial = 1;
for (i = 1; i <= n; i++)
{
  Factorial*=i;
  Suma +=i+1/ Factorial;
}
MessageBox.Show…
```

Приклад 5:
Знайти середнє геометричне парних чисел з проміжку [31; 76].

Приклад 6:
Обчислити 8^6 без використання функції піднесення до степеня.

Приклад 7:
Обчислити експоненту заданого числа, як суму членів ряду, припиняючи обчислення, коли черговий член за абсолютною величиною стане меншим 0.0001.

Тема 10. Обробка одновимірних масивів в C#

Масив – це сукупність однотипних елементів, яка має назву. Звертання до елементу масиву виконується за його індексом. В C# індекси елементів масиву починаються з нуля.

i	0	1	2	3	4	5	6
A	7	10	11	5	3	131	18

Для звертання до елементів масиву використовуються квадратні дужки [ ]. Наприклад, `i = 3; A[4]=3`.

При створенні масиву спочатку необхідно оголосити вказівку на нього як на масив, а після цього за допомогою оператора `new` виділити пам'ять під елементи масиву, вказавши тип даних. Наприклад:

```
int[] CountStudentGrup;
CountStudentGrup = new int [10];
або
Const int n = 15;
Double[] masA = new Double [N];
```

Приклад:

Для масиву з 10-ти елементів ввести його дійсні значення від користувача і знайти суму його елементів.

```
const int i, n = 10;
Double[] mas = new double[n];
Double, suma = 0;
String S;
for (i = 0; i < n; i++)
{
  if (!InputInt(ref mas[i], "Введіть " + i.ToString + "-й
елемент масиву"))
    return;
  suma +=mas[i];
}
```

Вивести сформований масив на екран:

```
rez ="";
for (i = 0; i < n; i++)
  rez += "mas[" + i.ToString() + "] = " + mas[i].ToString()
+ Strings.Chr(13);
MessageBox.Show(rez  + Strings.Chr(13) + Strings.Chr(13) +
"", "Результати обчислень", MessageBoxButtons.OK);
```

Значення елементів масиву можна і не вводити від користувача, а згенерувати за допомогою датчика випадкових чисел.

Приклад:

Згенерувати масив з n елементів, кожен з яких буде знаходитися в проміжку [a,b].

Для генерації масиву використаємо об'єкт класу `Random`, який використовують для генерації випадкових, цілих і дійсних чисел.

```
int i, n = 10;
Double a = 3, b = 17;
String rez;
if (!InputInt(ref n, "Введіть кількість елементів масиву"))
   return;
if (!InputDouble(ref a, "Введіть мінімальне значення"))
   return;
if (!InputDouble(ref b, "Введіть максимальне значення"))
   return;
Double[] mas = new double[n];
```

```
Random RND = new Random();
rez ="";
for (i = 0; i < n; i++)
{
   mas[i] = a + (b - a) * RND.NextDouble();
   rez   +=   "mas["   +   i.ToString()   +   "]   =   "   +
mas[i].ToString() + Strings.Chr(13);
}
MessageBox.Show(rez  + Strings.Chr(13) + Strings.Chr(13) +
"", "Результати обчислень", MessageBoxButtons.OK);
```

Приклад:

Визначити суму та добуток елементів масиву:

```
int i, n;
String S;
if (!InputInt(ref n, "Введіть кількість елементів масиву"))
  return;
Double[] mas = new double[n];
for (i = 0; i < n; i++)
  if  (!InputInt(ref  mas[i],  "Введіть  "  +  i.ToString  +
   "-й елемент масиву"))
   return;
Double Suma, Dobutok;
Suma = 0;
Dobutok = 1;
String rez;
for (i = 0; i < n; i++)
{
  Suma += mas[i];
  Dobutok *= mas[i];
}
S = "Для масиву: " + Strings.Chr(13);
for (i = 0; i < n; i++)
  S += "mas[" + i.ToString() + "] = " + mas[i].ToString() +
Strings.Chr(13);
  S   +=   "Добуток   рівний:   "   +   Dobutok.ToString()   +
Strings.Chr(13) + "Сума становить " + Suma.ToString();
MessageBox.Show(S   + Strings.Chr(13) + Strings.Chr(13) +
"", "Результати обчислень", MessageBoxButtons.OK);
```

Приклад:

Визначити мінімальний та максимальний елементи, кількість додатніх та від’ємних значень:

```
Min = mas[0];
Max = mas[0];
CountPlus = 0;
CountMinus = 0;
for (i = 0; i < n; i++)
{
   if (mas[i] > Max)
     Max = mas[i];
   if (mas[i] < Min)
     Min = mas[i];
   if (mas[i] > 0)
     CountPlus++;
   if (mas[i] < 0)
     CountMinus++;
```

```
}
```

Приклад:

Визначити кількість додатніх і відємних чисел серед елементів масиву.

```
CountV:=0;
CountD:=0;
For (i=0, i<N; i++)
 {if (mas[i]<0) CountV++;
  If (mas[i]>0) CountD++;
```

Приклад:

Знайти значення, найближче до середнього (n має бути >0):

```
Suma = 0;
for (i = 0; i < n; i++)
  Suma += mas[i];
SR = Suma/n;
VIDX = Math.Abs(mas[0] - SR);
index = 0; //індекс елементу найближчого до середнього
for (i = 0; i < n; i++)// оцінюємо відхилення інших елементів
  if (Math.Abs(mas[i] - SR) < VIDX)
  {
    VIDX = Math.Abs(mas[i] - SR);
    Index = i;
  }
...
```

Приклад:

Розподілити елементи масиву в два масиви так, щоб в першому масиві виявилися парні елементи, а в другому не парні:

```
CountParno = 0;
CountNeParno = 0;
for (i = 0; i < n; i++)
   if ((mas[i] % 2) == 0)
     CountParno++;
   else
     CountNeParno++;
Double[] masParno = new double[CountParno];
Double[] masNeParno = new double[CountNeParno];
indexParno = 0;
indexNeParno = 0;
for (i = 0; i < n; i++)
    if ((mas[i] % 2) == 0)
      masParno[indexParno++] = mas[i];
    else
      masNeParno[indexNeParno++] = mas[i];
```

Приклад:

Визначити першу позицію входження елемента в масив (задача пошуку елемента)

```
Index=0;
For (i=0, i<N; i++)
 If (mas[i]==element)
 {Index=I;
  Break; }
```

Тема 11. Сортування одновимірних масивів в C#

Приклад:

Написати програму для сортування елементів масиву за зростанням методами бульбашок, прямого включення, плямого вибору та швидкого сортування.

Наприклад, метод прямого включення полягає в :

a) відшуканні для чергового елемента позиції включення у вже відсортованій частині;
b) посуванні більших елементів;
c) включенні чергового елемента у звільнену позицію.

```
static void quickSort(double[] mas,int i,int j)
{   if (i >= j)
     return;
    double element = mas[i];
    int i1 = i; int j1 = j;
    while(i1<j1)
     {while (j1 > i1 && mas[j1] >= element)
       j1--;
      if(i1<j1)
       {mas[i1] = mas[j1];
        mas[j1] = element;
        i1++;
       }
      while (i1 < j1 && mas[i1] <= element)
       i1++;
      if (i1 < j1)
       {mas[j1] = mas[i1];
        mas[i1] = element;
        j1--;
       }
    }
    quickSort(mas, i, i1 - 1);
    quickSort(mas, i1 + 1, j);
}

static void generateMas(double[] mas, int N, double a, double b)
{
    Random rnd = new Random();
    for (int i = 0; i < N; i++)
        mas[i] = a + rnd.NextDouble() * (b - a);
}

static void printMas(double[] mas, int N, string s)
{
    Console.WriteLine(s);
    for (int i = 0; i < N; i++)
        Console.WriteLine("mas[" + i.ToString() + "]=" + mas[i].ToString());
}

static void Main(string[] args)
{
    const double a = 125, b = 13430;
    int N, i, j, k, indexMin;
    Console.Write("Введіть кількість елементів масиву: ");
    N = Convert.ToInt32(Console.ReadLine());
    double[] mas = new double[N];
    double element, min;

    generateMas(mas, N, a, b);
    printMas(mas, N, "Початковий массив");
    Console.ReadKey();
    //метод бульбашок
    DateTime startTime = DateTime.Now;
    for (i = 0; i < N - 1; i++)
     for (j = 0; j < N - i - 1;j++)
      if(mas[j]>mas[j+1])
       {element = mas[j];
```

```
    mas[j] = mas[j + 1];
    mas[j + 1] = element;
   }
DateTime finishTime = DateTime.Now;
printMas(mas, N, "\nВідсортований массив");
Console.WriteLine("Тривалість сортування методом бульбашок: " +
 (finishTime - startTime).ToString());
Console.ReadKey();

generateMas(mas, N, a, b);
printMas(mas, N, "Початковий массив");
Console.ReadKey();
//метод прямого включення
startTime = DateTime.Now;
for (i = 1; i < N;i++ )
{   j = 0;
    element=mas[i];
    while (j < i && mas[j] <= mas[i])
        j++;
    if(j<i)
    {for (k = i - 1; k >= j; k--)
      mas[k + 1] = mas[k];
     mas[j] = element;
    }
}
finishTime = DateTime.Now;
printMas(mas, N,"\nВідсортований массив");
Console.WriteLine("Тривалість сортування методом прямого включення: " +
 (finishTime - startTime).ToString());
Console.ReadKey();

generateMas(mas, N, a, b);
printMas(mas, N, "Початковий массив");
Console.ReadKey();
//метод прямого вибору
startTime = DateTime.Now;
for (i = 0; i < N-1; i++)
{
    min = mas[i];
    indexMin = i;
    for (j = i + 1; j < N; j++)
     if (mas[j]<min)
      {min = mas[j];
       indexMin=j;
      }
    if (indexMin>i)
     {mas[indexMin] = mas[i];
      mas[i] = min;
     }
}
finishTime = DateTime.Now;
printMas(mas, N, "\nВідсортований массив");
Console.WriteLine("Тривалість сортування методом прямого вибору: " +
 (finishTime - startTime).ToString());
Console.ReadKey();

generateMas(mas, N, a, b);
printMas(mas, N, "Початковий массив");
Console.ReadKey();
//метод швидкого сортування обміном на великих відстанях
startTime = DateTime.Now;
quickSort(mas, 0, N - 1);
finishTime = DateTime.Now;
printMas(mas, N, "\nВідсортований массив");
Console.WriteLine("Тривалість швидкого сортування методом на великих відстанях:"+
 (finishTime - startTime).ToString());
Console.ReadKey();
```

```
        generateMas(mas, N, a, b);
        printMas(mas, N, "Початковий массив");
        Console.ReadKey();
        //стандартний метод сортування
        startTime = DateTime.Now;
        Array.Sort(mas);
        finishTime = DateTime.Now;
        printMas(mas, N, "\nВідсортований массив");
        Console.WriteLine("Тривалість стандартного сортування: "
         + (finishTime - startTime).ToString());
        Console.ReadKey();
    }
```

Тема 12. Двовимірні масиви в C#

В мовах програмування часто доводиться обробляти не лише одновимірні, а й багатовимірні масиви. В процесі опису виміри масивів перераховуються через кому, а для обробки таких масивів найчастіше використовуються вкладені цикли з параметрами по різних змінних. Наприклад, для опису матриці m*n вказують:

Const M=10, N=12;

int [,] Mas = new int[M, N];

Приклад:

Написати програму для множення квадратичних матриць (двовимірних масивів)

```
For (i=0; i<N; i++) //цикл по рядках матриці добутку
 For (j=0; j<N; j++) //цикл по стовбцях
  {Suma=0;
   For (k=0; k<M; k++)
    Suma+= A[i,k]*B[k,j];
   C[i;j]=suma;
  }
//виводимо елементи матриці С в командному рядку
For (i=0; i<N; i++) //цикл по рядках матриці добутку
 {For (j=0; j<N; j++) //цикл по стовбцях
   Console.Write(C[i,j].ToString()+"\t"); }
  Console.WriteLine();
 }
Console.ReadLine();
```

Тема 13. Обробка рядків в програмуванні

1. Символи та рядки в C#. Принципи порівняння чисел і рядків.
2. Алгоритми обробки рядків.
3. Алгоритми обробки слів.
4. Стандартні функції обробки рядків.

1. Для обробки рядків в C# використовуємо тип даних **String**. Фактично це послідовність символів в кодуванні **UTF8**. В цьому універсальному кодуванні, для коду кожного символа в середньому відводиться 2 байти.

Фактично String – це масив символів, але при порівнянні рядків порівнюють не адреси масивів, а коди окремих символів. Для обробки окремих символів в C# використовується структура char – це код одного символа. Текстові константи беруться в подвійні лапки, а окремі символи – в одинарні.

```
Char C;
C = ‘E’;
C = ‘K’;
```

В символах можна безпосередньо записати його код, виконавши явне перетворення, - це присвоїти: **C=(char)65**, що еквівалентно **C = ‘A’ ;**

В рядках символів допускається використання **Escape**, які задаються або спеціальними знаками, або кодами символів після \, наприклад, “\n”. Для ігнорування Escape символів спереду передм лапками встановлюється **@**.

```
String S = @ “D:\ЕК21\”
```

Рядки між собою порівнюються з ліва на право по кодах символів, а числа порівнюються по значеннях.

Тому, якщо `S1 = “25” ;`
`S2 = “137”;`
`S1 > S2,` але `Convert.ToInt32(S1)<Convert.ToInt32(S2)`

2. Для визначення довжини рядка використовується його властивість – **Length.** Рекомендується значення цієї властивості записувати в локальну змінну для прискорення звертання до цього значення і пришвидчення роботи програми.

Зчитування коду з окремої позиції рядка виконується за її індексом:

S [0] – код першого символа.

S [S.Length-1] – код останнього символа**.**

Для отримання ж рядка з першого символа необхідно явно перетворити його код в рядок: **S[0].ToString();**

Завдання 1: Скласти програму, яка перепише текстовий рядок навпаки

```
String S1, S2;
S1 = Interaction.InputBox ("Ведіть рядок для обробки", "Введення", "Значення по замовчуванню");
S2 = "";
int I, len;
len = S1.Length;
for (int i =len -1 ; i >=0 ; i --)
S2 += S1[i].ToString();
MessageBox.Show("При обробці рядка '" + S1 + "'отримано рядок+'  "  +  S2  +  "'",  "  результати",  MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Інший варіант реалізації – брати символи з ліва на право, але дописувати їх спереду

```
    for (int i =0 ; i < len ; i ++)
         S2 = S1[i].ToString()+S2;
```

Завдання 2: Вилучити з рядка пробіли.

```
String S1, S2;
```

```
    S1 = Interaction.InputBox("Ведіть рядок для обробки",
"Введення", "Значення по замовчуванню");
    S2 = "";
    int I, len;
    len = S1.Length;
    for (int i = 0; i < len; i++)
        if (S1 [i].ToString()!= " ")
            S2 += S1[i].ToString();
      MessageBox.Show("При обробці рядка '" + S1 + "'отримано
  рядок+' " + S2 + "'", " результати", MessageBoxButtons.OK,
  MessageBoxIcon.Information);
```

Завдання 3: Підрахувати кількість входжень певної букви.

Завдання 4: Вставити після кожної букви кому.

Завдання 5: Визначити позиції входження ‘к’.

Завдання 6: Замінити буквосполучення «Оце» на «це».

```
String S, S1;
<Ввести S>
Int len=S.Length;
S1="";
If (len<3)
 S1=S
Else
{ for(i=0; i<len-2; i++)
   If (S[i]<>'0 | S[i+1]<>'ц' | S[i+2]<>'е')
    S1+=S[i].ToString();
  S1+=S[len-2].toString()+S[len-1].toString();
}
<Вивести S1>;
```

Завдання 7: Вилучити з рядка зайві пробіли.

Завдання 8: Встановити, чи введений рядок – паліндром.

Завдання 9: Замінити ‘.’ на «…» і навпаки.

3. Завдання 10: Скласти програму для поділу введеного рядка на окремі слова. Вважатимемо словами послідовності символів, які йдуть між розділовими знаками. В рядку послідовно будемо шукати початок слова (індексі **«i»)** і перший символ за словом (індексі **«j»**) до тих пір поки не обробимо весь рядок. Тому і будемо збільшувати поки йдуть розділові знаки, а **«j»** необхідно починати збільшувати з **i+1** доки не зустрінемо розділовий знак або кінець рядка. Для переходу до наступного слова **i= j+1** і цикл повторюється знову.

```
Static Bool RozdilZnak (Char C)
{ If (C==' ' | C==',’| C=='-'| C=='.'…)
        Return True;
  Else
        Return False;
}
```

Реалізація процедури:

```
String S, S1="";
S= …;
int i, len =S.Length;
i=0;
While (i<len)
```

```
    {while (i<len && RozdilZnak((char)S[i]))
      i++;
    If (i<len)
     { j=i+1;
       While (j<len && !RozdilZnak((char)S[j]))
        j++;
       S1+=S.Substring(i, j-i) + "\r\n";
       i=j+1;
     }
  }
     <Вивести S1>
```

Завдання 11: Визначити кількість слів, в яких перший і останній символи співпадають.

Завдання 12: В словах рядка замінити закінчення *ий* на *еньский*.

Завдання 13 (конкурс): Визначити, з якої букви починається найбільше слів у рядку.

4. Стандартні функції для обробки рядків в C#:

– **S.SubString**(<позиція початку>, <кількість>) – вирізає з заданої позиції рядка вказану кількість символів.

– **String.Compare**(<STR1>, <STR2>, <bool>). Статична функція, що повертає одиницю, якщо перший рядок більший за другий, 0 – якщо рівні, -1 –якщо 2-ий рядок більший за 1-й. Якщо останній аргумент істинний, то порівняння відбувається з врахуванням регістру, інакше – без врахування.

– **String.Equals**(<STR1>, <STR2>). Повертає «істинно», якщо рядки рівні, і «хибно» в протилежному випадку.

– **S.Insert**(<позиція>, <рядок>). Вставляє рядок з вказаної позиції. Викликається як метод для рядка. Повертає новий рядок.

– **IndexOf**(<підрядок>). Виконує пошук підрядка в рядку. Якщо пошук результативний повертає номер першої віднайденої позиції, інакше повертає (-1).

– **ToUpper**(). Переводить рядок до верхнього регістру.

– **ToLower()**. Переводить рядок до нижього регістру.

В процесі модифікації рядків кожного разу C# створює новий рядок в динамічній пам'яті, а старий залишає для знищення збірнику сміття. Тому у випадках інтенсивної обробки рядків замість класу String доцільно використовувати клас **StringBuilder.**

Наприклад:

Створити програму, яка кожну першу букву слів в рядку переводить до верхнього регістру.

Процедура обробки події натиснення кнопки:

```
…
j=o, i=o //починаємо обробку рядка;//
while (i<len)
 { while (i<len && RozdilZnak((char)S[i]))
    i++;
   If(i>j)
    Rez+= S.Substring (j, i-j);
   If(i<len)
    { j=i+1;
      While (j<len && !RozdilZnak(char)S[j]))
       j++;
      Rez+= (S[i].ToString()).ToUpper();//дописати першу букву до
верхнього регістру.//
      If((j-i)>1) //в черговому слові більше однієї букви//
       Rez+= S.Substring (j+1, j-i-1);
      i=j;
   }}…
```

Тема 14. Структури та їх застосування для обробки структурованих даних

1. Синтаксис опису структур.

Структура – це тип **даних значень**, що поєднує в собі дані інших типів та засоби для їх обробки. Оголошення структури здійснюється за допомогою ключового слова *struct*. **Структури не можуть реалізовувати наслідування, але можуть підтримувати інтерфейси.**

Синтаксис опису структури:

```
[<модифікатор доступу>] struct [ім'я_структури]
{       тип1 елемент1;
        тип2 елемент2;
        ........................
        типN елементN;
  [<засоби обробки даних структури (конструктори, методи,
   властивості, індексатори…)>]
};
```

Приклад структури даних книги:

```
struct Book
{   public string name;
    public string author;
    public int year;
    public void Info()
    {Console.WriteLine("Книга '{0}' (автор(и) {1}) була видана в {2} році",
      name, author, year); }}
```

Пам'ять у структурі розподіляється покомпонентно, зліва направо, від молодших до старших адрес пам'яті:

```
struct dataTypes {
 double aFloat;
 int anInt;
 string aString;
 char aChar;
 char aLong;
};
```

Потрібно відзначити, що опис структури не виділяє місце у пам'яті під елементи структури. Її опис визначає лише так званий шаблон, який описує характеристики змінних, що будуть розміщуватися у конкретній структурі. Щоб ввести змінні та зарезервувати для них пам'ять необхідно окремо оголосити змінні типу структури. Доступ до окремого елемента структури забезпечується операторами вибору: . (прямий селектор), наприклад:

```
Book book;
book.name = "Изучаем C#";
book.author = "Ендрю Стиллмен, Дженіффер Грин";
book.year = 2017;
//Виводимо інформацію про книгу
book.Info();
```

Для змінних одного і того ж самого структурного типу, як і для довільного типу значення, визначена операція присвоєння, при цьому здійснюється **поелементне копіювання значень полів**.

Але, для порівняння структур необхідно перевіряти рівність відповідних полів цих структур.

```
struct point
{double x,y;
  char c; }
…
point  point1,point2;
if (point1.x==point2.x && point1.y==point2.y &&
    point1.c==point2.c)
{ };
```

Кожний опис структури вводить унікальний тип структури, тому в наступному фрагменті програми об'єкти *a* і *a1* мають однаковий тип *struct A*, але об'єкти *a* і *b* мають різні типи структури:

```
struct A {
 int i,j;
 double d; };
struct B {
 int i,j;
 double d; };
…
A a, a1;
B b;
```

Структурам можна виконувати присвоювання тільки в тому випадку, якщо і вихідна структура, і структура, в яку виконується присвоєння, мають один і той же тип:

```
a = a1; /*можливо, оскільки a і a1 мають однаковий тип*/
a = b; /* неможливо, бо змінні належать до різних типів*/
```

2. Ініціалізація структур.

Перед використанням структур чи викликом їх методів всі їхні дані мають бути проініціалізовані, оскільки структури – це типи значень. Це можна зробити за допомогою:

d) послідовного занесення значень в поля, як було показано вище;

e) виклику конструктора по замовчуванню, який занесе в поля значення по замовчуванню. Наприклад, *Book book = new Book()*;

f) виклику конструктора з параметрами, який занесе в поля передані значення параметрів або введе їх від користувача. Для полів, не заповнених конструктором, встановлюються значення по замовчуванню. Наприклад, *Book book = new Book(“Кобзар”, «Шевченко», 1841)*;

g) ініціалізатора, який описується в фігурних дужках після конструктора. В тексті ініціалізатора перераховуються через кому поля з присвоєнням значень, які в них заносяться. Непроініціалізовані поля заповнюються значеннями по замовчуванню. Наприклад, *Book book1 = new Book{name = "Кобзар", author = "Шевченко"};*

3. Приклад використання структур.

```
class Program
    {
     static void Main(string[] args)
      {Book book;
       //ініціалізація способом послідовного занесення
       book.name = "Изучаем C#";
       book.author = "Єндрю Стиллмен, Дженіффер Грин";
       book.year = 2017;
       book.Info(); //Виводимо інформацію про книгу

       //Використання ініціалізатора після конструктора по замовчуванню
       Book book1 = new Book{name="Изучаем C#", author="Єндрю Стиллмен, Дженіффер Грин"};
       //Рік по замовчуванню ініціалізується 0
```

```
    book1.Info();

    //Виклик конструктора з параметрами. Рік ініціалізується в конструкторі
    Book book2 = new Book("A", "B", 1970) { name = "Изучаем C#",
     author = "Ендрю Стиллмен, Дженнифер Грин" };
    book2.Info();

    //Формування масиву з чотирьох кних і вибір серед них найсвіжішої
    Book[] masBook = new Book[]
     { new Book {name = "Изучаем C#", author = "Ендрю Стиллмен, Дженнифер Грин"},
       new Book("Кобзар", "Тарас Григорович Шевченко", 1840),
       new Book(true), new Book(false), new Book()};
       Console.WriteLine("Перелік книг:");
       foreach (Book cBook in masBook)
        cBook.Info();
       int index = 0, lastYear=masBook[0].year;
       for (int i = 1; i < masBook.Count(); i++ )
        if (masBook[i].year > lastYear)
         {lastYear=masBook[i].year;
          index=i; }
       Console.Write("Серед них найсвіжіша: ");
       masBook[index].Info();
       Console.ReadLine(); }

struct Book
 {public string name;
  public string author;
  public int year;

  public Book(bool readData):this()
  {if (readData)
    {Console.Write("Введіть назву: ");
     this.name = Console.ReadLine();
     Console.Write("Введіть автора: ");
     this.author = Console.ReadLine();
     Console.Write("Введіть рік виходу: ");
     this.year = Convert.ToInt32(Console.ReadLine()); }}

     public Book(string name, string author, int year)
     {this.name =name;
      this.author=author;
      this.year=year; }

     public void Info()
     {Console.WriteLine("Книга '{0}' (автор(и) {1}) була видана в {2} році", name,
        author, year); }
 }
```

Тема 15. Обробка файлів як цілісних об'єктів

Файл – це послідовність байтів, що зберігається на зовнішньому пристрої (наприклад, на жорсткому диску) та має назву. Скорочена назва файла містить його ім'я та розширення. Саме розширення дозволяє ОС ідентифікувати, які дані і в якому форматі зберігаються у файлі. Повна назва файла містить букву диску, перелік вкладених каталогів та скорочену назву. При роботі з файлами для однозначної ідентифікації обов'язково вказується їх повна чи скорочена назва. Використання скороченої назви передбачає обробку файла в поточному каталозі.

Для роботи з файлами, як цілісними об'єктами, необхідно:

1. Відкрити простір імен `System.IO`;
2. Використовувати методи статичного класу `File`.

Під роботою з файлами мається на увазі:
- Створення файлів;
- Видалення файлів;
- Читання даних;
- Запис даних;
- Зміна параметрів файлу (ім'я, розширення ...);
- Інше.

У C# є простір імен System.IO, в якому реалізовані всі необхідні нам класи для роботи з файлами. Щоб підключити цей простір імен, необхідно на самому початку програми додати рядок using System.IO.
Сумісно можна використовувати простір імен System.Text, що допоможе використовувати різні кодування тексту.

Як створити файл?

Для створення порожнього файлу, в класі File є метод Create (). Він приймає один аргумент - шлях. Нижче наведений приклад створення порожнього текстового файлу new_file.txt на диску D. Якщо файл з таким ім'ям вже існує, він буде переписаний на новий порожній файл.

```
using System;
using System.Text;
using System.IO;
    class MyPage
    {
    static void Main()
       {
         File.Create("D:\\new_file.txt");
       }
     }
```

Метод WriteAllText () створює новий файл (якщо такого немає), або відкриває існуючий і записує текст, замінюючи все, що було у файлі:

```
using System;
using System.Text;
using System.IO;
    class MyPage
    {
    static void Main()
       {
         File. WriteAllText("D:\\new_file.txt", "текст");
       }
     }
```

Метод AppendAllText () працює, як і метод WriteAllText () за винятком того, що новий текст дописується в кінець файлу, а не переписує все що було у файлі:

```
    static void Main(string[] args)
       {
         File.AppendAllText("D:\\new_file.txt", "тест метода AppendAllText ()"); //допише текст в кінець файлу
       }
```

Як видалити файл?

Метод Delete () видаляє файл за вказаним шляхом:

```
static void Main()
{
   File.Delete("d:\\test.txt"); //видалення файлу
}
```

Потоки

Крім того, щоб читати / записувати дані у файл з Сі-шарп можна використовувати потоки.
Потік - це абстрактне уявлення даних (у байтах), яке полегшує роботу з ними. В якості джерела даних може бути файл, пристрій введення-виведення, принтер.
Клас Stream є абстрактним базовим класом для всіх потокових класів у Сі-шарп. Для роботи з файлами нам знадобиться класFileStream (файловий потік).
FileStream - представляє потік, який дозволяє виконувати операції читання / запису у файл.

```
static void Main()
{
   FileStream file = new FileStream("d:\\test.txt", FileMode.Open, FileAccess.Read); // відкриває файл тільки на читання
}
```

Режими відкриття FileMode:

- Append - відкриває файл (якщо той існує) і переводить покажчик у кінець файлу (дані будуть дописуватися до кінця), або створює новий файл. Даний режим можливий тільки при режимі доступу FileAccess.Write.
- Create - створює новий файл (якщо той існує - замінює його)
- CreateNew - створює новий файл (якщо той існує - генерується "винятки")
- Open - відкриває файл (якщо той не існує - генерується "винятки")
- OpenOrCreate - відкриває файл, або створює новий, якщо його не існує
- Truncate - відкриває файл, але всі дані всередині файлу затирає (якщо файлу не існує - генерується "винятки")

```
static void Main ()
{
    FileStream file1 = new FileStream ("d: \ \ file1.txt", FileMode.CreateNew); / / створення нового файлу
    FileStream file2 = new FileStream ("d: \ \ file2.txt", FileMode.Open); / / відкриття існуючого файлу
    FileStream file3 = new FileStream ("d: \ \ file3.txt", FileMode.Append); / / Відкриття файлу на дозапис в кінець файлу
}
```

Режими доступу FileAccess:
- Read - відкриття файлу тільки на читання. При спробі запису генерується виключення
- Write - відкриття файлу тільки на запис. При спробі читання генерується виключення
- ReadWrite - відкриття файлу на читання і запис.

Читання з файлу

Для читання даних з потоку нам знадобитися клас StreamReader. У ньому реалізовано безліч методів для зручного зчитування даних. Нижче наведена програма, яка виводить вміст файлу на екран:

```
static void Main ()
{
    FileStream file1 = new FileStream ("d:\\test.txt", FileMode.Open); // створюємо файловий потік
    StreamReader reader = new StreamReader (file1); // створюємо «потоковий читач» і пов'язуємо його з файловим
потоком
    Console.WriteLine (reader.ReadToEnd ()); // зчитуємо всі дані з потоку і виводимо на екран
    reader.Close (); // закриваємо потік
    Console.ReadLine ();
}
```

Метод ReadToEnd () зчитує всі дані з файлу. ReadLine () - зчитує один рядок (покажчик потоку при цьому переходить на новий рядок, і при наступному виклику методу буде лічена наступна рядок).
Властивість EndOfStream вказує, чи знаходиться поточна позиція в потоці в кінці потоку (чи досягнуто кінець файлу). Повертає true абоfalse.

Запис у файл

Для запису даних у потік використовується клас StreamWriter. Приклад запису у файл:

```
static void Main ()
{
  FileStream file1 = new FileStream ("d:\\test.txt", FileMode.Create); // створюємо файловий потік
  StreamWriter writer = new StreamWriter (file1); // створюємо «потоковий письменник» і пов'язуємо його з файловим
потоком
  writer.Write ("текст"); // записуємо в файл
  writer.Close (); // закриваємо потік. Чи не закривши потік, у файл нічого не запишеться
}
```

Метод WriteLine () записує у файл порядково (те ж саме, що і простий запис за допомогою Write (), тільки в кінці додається новий рядок).
Потрібно завжди пам'ятати, що після роботи з потоком, його треба закрити (звільнити ресурси), використавши метод Close().

Кодування, в якій будуть зчитуватися / записуватися дані вказується при створенні StreamReader / StreamWriter:

```
static void Main ()
{
   FileStream file1 = new FileStream ("d:\\test.txt", FileMode.Open);
   StreamReader reader = new StreamReader (file1, Encoding.Unicode);
   StreamWriter writer = new StreamWriter (file1, Encoding.UTF8);
}
```

Крім того, при використанні StreamReader і StreamWriter можна не створювати окремо файловий потік FileStream, а зробити це відразу при створенні StreamReader / StreamWriter:

```
static void Main ()
{
   StreamWriter writer = new StreamWriter ("d:\\test.txt"); / / вказуємо шлях до файлу, а не потік
   writer.WriteLine ("текст");
   writer.Close ();
}
```

Як створити папку?

За допомогою статичного методу CreateDirectory() класу Directory:

```
static void Main ()
{
  Directory.CreateDirectory ("d:\\new_folder");
}
```

Як видалити папку?

Для видалення папок використовується метод Delete():

```
static void Main ()
{
   Directory.Delete ("d:\\new_folder"); / / видалення порожньої папки
}
```

Якщо папка не порожня, необхідно вказати параметр рекурсивного видалення - true:

```
static void Main ()
{
   Directory.Delete ("d:\\new_folder", true); / / видалення папки, і всього, що всередині
}
```

Тема 16. Текстові файли та потоки

Файл — это упорядоченная и именованная последовательность байтов, имеющая постоянное хранилище. При работе с файлами используются пути к каталогам, запоминающие устройства, а также имена файлов и каталогов. В отличие от файла, поток — это последовательность байтов, которую можно использовать для записи или чтения из вспомогательного запоминающего устройства, являющегося одним из устройств хранения информации (например, дисков или памяти). Есть несколько видов запоминающих устройств, отличных от дисков, и существует несколько видов потоков, помимо файловых потоков, например сетевые потоки, потоки памяти и потоки каналов.

Потоки

Абстрактный базовый класс Stream поддерживает чтение и запись байтов. Все классы, представляющие потоки, являются производными от класса Stream. Класс Stream и его производные классы обеспечивают общий способ просмотра источников данных и хранилищ объектов, а также изолируют программиста от специфических особенностей операционной системы и базовых устройств.

Потоки включают три основные операции:

- Чтение — перенос информации из потока в структуру данных, такую как массив байтов.
- Запись — перенос данных в поток из источника данных.
- Поиск — определение и изменение текущей позиции внутри потока.

В зависимости от базового источника или хранилища данных поток может поддерживать лишь некоторые из этих возможностей. Например, класс PipeStreamне поддерживает поиск. Свойства CanRead, CanWrite и CanSeek потока определяют операции, поддерживаемые потоком.

Ниже перечислены некоторые часто используемые классы потока:

- FileStream — для чтения и записи в файл.
- IsolatedStorageFileStream — для чтения и записи в файл в изолированном хранилище.
- MemoryStream — для чтения и записи в память в качестве резервного хранилища.
- BufferedStream — для повышения быстродействия операций чтения и записи.
- NetworkStream — для чтения и записи на сетевые сокеты.
- PipeStream — для чтения и записи в анонимные и именованные каналы.
- CryptoStream — для связи потоков данных с криптографическими преобразованиями.

Пример асинхронной работы с потоками см. в разделе Асинхронный файловый ввод-вывод.

Средства чтения и записи

Пространство имен System.IO также предоставляет типы для чтения закодированных символов из потоков и их записи в потоки. Как правило, потоки предназначены для ввода и вывода байтов. Типы чтения и записи обрабатывают преобразование закодированных символов в байты или из байтов, чтобы поток мог завершить операцию. Каждый класс чтения и записи связан с потоком, который можно получить с помощью свойства класса `BaseStream`.

Ниже перечислены некоторые часто используемые классы для чтения и записи:

- 1. BinaryReader и BinaryWriter — для чтения и записи простых типов данных, таких как двоичные значения.
- 2.1. StreamReader и StreamWriter — для чтения и записи символов с использованием закодированного значения для преобразования символов в байты или из байтов.
- 2.2. StringReader и StringWriter — для чтения и записи символов в строки или из строк.
- 2. TextReader и TextWriter используются в качестве абстрактных базовых классов для других средств чтения и записи, которые считывают и записывают символы и строки, а не двоичные данные.

```
//створення форматованого текстового файла з числами
            StreamWriter f = new StreamWriter("I:\\LR18Num.txt");//+bool append +Encoding
            int i; char[] mc = new char[3];
```

```
            for (i = 0; i < 100; i++)
                f.Write("{0,3}",i);
            f.Close();
//обробка форматованого текстового файла з числами
            StreamReader f1 = new StreamReader("I:\\LR18Num.txt");
            while (!f1.EndOfStream)
            {f1.Read(mc, 0, 3);
             string s = new string(mc);
             i=Convert.ToInt32(s);
             Console.WriteLine(i.ToString());
            }
            f1.Close();
            Console.ReadLine();
//створення текстового файла
            StreamWriter f = new StreamWriter("I:\\LR18.txt",);
            int i;
            for (i = 0; i < masBook.Count(); i++)
            {
                f.WriteLine(masBook[i].author);
                f.WriteLine(masBook[i].name);
                f.WriteLine(masBook[i].year.ToString());
            }
            f.Close();

//обробка текстового файла
            StreamReader f1 = new StreamReader("I:\\LR18.txt");
            int countLine=0;
            while (!f1.EndOfStream)
            {f1.ReadLine();
             countLine++;
            }
            if (countLine==0)
            {
             Console.WriteLine("Дані книг відсутні");
             Console.ReadLine();
            }
    f1.BaseStream.Seek(0,SeekOrigin.Begin); //позиціонуємо на початок в базовому потоці
    f1.DiscardBufferedData();//очищаємо проміжний буфер для коректного зчитування з початку
            masBook = new Book[countLine / 3];
            i = 0;
            while (!f1.EndOfStream)
            {masBook[i].author=f1.ReadLine();
             masBook[i].name = f1.ReadLine();
             masBook[i].year = Int32.Parse(f1.ReadLine());
             i++;
            }
            f1.Close();

            Console.WriteLine("Перелік книг:");
            foreach (Book cBook in masBook)
                cBook.Info();
            int index = 0, lastYear = masBook[0].year;
            for (i = 1; i < masBook.Count(); i++)
                if (masBook[i].year > lastYear)
                {
                    lastYear = masBook[i].year;
                    index = i;
                }
            Console.Write("Серед них найсвіжіша: ");
            masBook[index].Info();
            Console.ReadLine();
        }
```

Тема 17. Двійкові файли та потоки

Для зчитування/запису двійкових даних у файл необхідно спочатку створити файловий потік, а потім оголошувати для нього `BinaryWriter` чи `BinaryReader` з можливістю задання кодування та повторного використання.

```
f = new FileStream("I:\\LR19.dat", FileMode.Create);
BinaryWriter w = new BinaryWriter(f , Encoding.UTF8);
int i;
for (i = 0; i < masBook.Count(); i++)
{   w.Write(masBook[i].author+"");
    Console.WriteLine(w.BaseStream.Position);
    w.Write(masBook[i].name+"");
    Console.WriteLine(w.BaseStream.Position);
    w.Write(masBook[i].year);
    Console.WriteLine(w.BaseStream.Position);
}
w.Close();

f = new FileStream("I:\\LR19.dat", FileMode.OpenOrCreate);
BinaryReader r = new BinaryReader(f , Encoding.UTF8);
int countZap=0;
while (r.BaseStream.Position<r.BaseStream.Length) //або r.Peek()>-1
{r.ReadString();
 r.ReadString();
 r.ReadInt32();
 countZap++;
}
if (countZap == 0)
{Console.WriteLine("Дані книг відсутні");
 Console.ReadLine();
}
r.BaseStream.Seek(0,SeekOrigin.Begin); //позиціонуємо на початок в базовому потоці
masBook = new Book[countZap];
i = 0;
while (i < countZap)
{masBook[i].author=r.ReadString();
 masBook[i].name = r.ReadString();
 masBook[i].year = r.ReadInt32();
 i++;
}
r.Close();

Console.WriteLine("Перелік книг:");
foreach (Book cBook in masBook)
    cBook.Info();
int index = 0, lastYear = masBook[0].year;
for (i = 1; i < masBook.Count(); i++)
    if (masBook[i].year > lastYear)
    {lastYear = masBook[i].year;
     index = i;
    }
Console.Write("Серед них найсвіжіша: ");
masBook[index].Info();
Console.ReadLine();
```

Тема 18. Зберігання та обробка у файлах структурованих даних

Для зберігання відомостей про об'єкти предметної області та їх взаємодію на сьогодні найчастіше використовуються таблиці БД. Рядки цих таблиць називають *записами*, а стовпці – *полями*. Основна частина даних в цих таблицях структурована, тобто кожне *поле* має визначений тип даних та довжину. Така структуризація дає змогу виконати швидкий доступ до даних довільного поля будь-якого запису, змістившись від початку даних на

$$offset=(i-1)*len+zm$$

байтів (де i – номер запису, len – довжина запису в байтах, zm – зміщення поля в середині запису) та зчитавши визначену для довжини цього поля кількість байтів. Тобто структуризація дозволяє здійснювати швидкий доступ до довільних даних, не перечитуючи попередні дані (як у текстових чи двійкових файлах послідовного доступу), хоча й призводить до зайвих витрат пам'яті.

Програміст може скористатися перевагами довільного доступу до даних файлів і потоків, якщо забезпечить зберігання даних в них згідно визначеної структури, тобто зберігання для кожного поля визначеної кількості байтів.

Типи значень C# (наприклад, числові чи логічні поля) мають визначену кількість байтів довжини відповідно своєму типу даних. Для типів посилань забезпечити відповідність даних визначеній структурі має програміст. Наприклад, для рядків необхідно обрати кодування, яке має фіксовану кількість байтів для кожного символа (ми використаємо Unicode), доповнювати їх визначеними символами (наприклад, пробілами) до потрібної довжити, якщо вони закороткі і обрізати – якщо задовгі, та врахувати, що перед символами рядка в потоці ще збегігається байт кількості цих символів. Наприклад, для зберігання авторів чи назви книги достатньо 50 символів, тому в кодуванні Unicode для цип полів потрібно відвести 101 байт.

Приклад. Написати програму для запису у файл структурованих даних про книги.

```
const string plusStr = "                                                  ";
int lenZap = 206;
FileStream f = new FileStream("I:\\LR19.dat", FileMode.Create);
BinaryWriter w = new BinaryWriter(f, Encoding.Unicode);
int i;
for (i = 0; i < masBook.Count(); i++)
 {w.Write((masBook[i].author+plusStr).Substring(0,50));
  Console.WriteLine(w.BaseStream.Position);
  w.Write((masBook[i].name+plusStr).Substring(0,50));
  Console.WriteLine(w.BaseStream.Position);
  w.Write(masBook[i].year);
  Console.WriteLine(w.BaseStream.Position);
 }
w.Close();
```

Приклад. Написати програму для зчитування з структурованого файлу даних про найсвіжішу книгу.

```
int lenZap = 206;
FileStream f = new FileStream("I:\\LR19.dat", FileMode.OpenOrCreate);
BinaryReader r = new BinaryReader(f,Encoding.Unicode);
int countZap=(int)r.BaseStream.Length/lenZap;
if (countZap == 0)
{Console.WriteLine("Дані книг відсутні");
 Console.ReadLine();
}
r.BaseStream.Seek(202,SeekOrigin.Begin); //позиціонуємо на початок в базовому потоці
int index = 0, year, lastYear = r.ReadInt32();
for (i = 1; i < countZap; i++)
 {//позиціонуємо на початок даних року активного запису в базовому потоці
  r.BaseStream.Seek(i*lenZap+202,SeekOrigin.Begin);
  year = r.ReadInt32();
  if (year > lastYear)
  {lastYear = year;
   index = i;
  }
}
```

```
//зчитуємо дані найсвіжішої книги
r.BaseStream.Seek(index*lenZap,SeekOrigin.Begin);
book.author=r.ReadString().TrimEnd();
book.name = r.ReadString().TrimEnd();
book.year = r.ReadInt32();
r.Close();
Console.Write("Дані найсвіжішої книги: ");
book.Info();
Console.ReadLine();
```

Тема 19. Створення власних методів та бібліотек класів

1. **Перезавантажувані методи. Створення власних методів для виконання повторюваних операцій.**
2. **Розробка і використання власних бібліотек класів**

1. Власні методи, як правило, розробляються для виконання повторюваних операцій.
Приклад 1. Обчислити значення виразу:

$$\frac{x^y + x^z}{z^x}$$

```
        Double x, y,  z, rez;
        x = Convert.ToDouble(poleX.Text);
        y = Convert.ToDouble(poleY.Text);
        z = Convert.ToDouble(poleZ.Text);
            rez = (Math.Pow(x , y) +  Math.Pow(x. z) / Math.Pow( z,
        x);
        PoleRez.Text = rez.ToString();
}
```

Забезпечення коректності введення чисел у текстові поля необхідно виконувати як в процесі введення так і перед обчисленням. Для обчислень необхідно вводити x, y, z, тому щоб не писати програму контролю 3 рази створимо підпрограму і будемо її викликати 3-чі. На відмінну від консольного додатку, у випадку введення **не числа** необхідно вивести повідомлення і забезпечити перехід курсора у поле, де введено **не число.**

Результати виконання функції є логічне значення, оскільки користувач може число ввести чи не ввести. У випадку введення числа ф-ція має також повернути його у місце виклику, тому сама змінна має передаватися у ф-цію з модифікатором **Ref.**

Ця ф-ція має бути універсальною, тобто необхідно забезпечити можливість її виклику ззовні навіть коли конкретного екземпляру (форми) не існує. Тому опишемоі-цію з модифікатором **Static.**

```
     Static Bool ConvertToDouble
     (Ref Double zminna, StringText, String.Povidom)
     {Try //відключити контроль помилок//
       {zminna=Convert.ToDouble(Text);
       }
     Catch(System.FormatExeption) // опрацювання помилок//
     MessageBox.Show(Povidom,    «Увага»,    MessageBoxButtons.Ok,
MessageBoxIcon.Error);
     Return False;
     Return True;
     }
```

Приклад 2: Створити процедуру обробки події, яка буде обчислювати значення виразу:

$$S = \frac{\sqrt{4^2 + x^2}}{\sqrt{x^2 + y^2}} + \frac{\sqrt{65^2 + y^2}}{\sqrt{6{,}4^2 + z^2}}$$

```
  Double x=0, y=0, z=0, rez;
  If(ConvertToDouble(x, poleX.Text, «Ви ввели не число в поле x»))
  {
   PoleX.SetFocus(); //перевести курсор//
   Return;
  }
  If(!ConvertToDouble(y, poleY.Text, «Ви ввели не число в поле
  y»))
  {
   PoleY.SetFocus();
```

```
 Return;
}
If(!ConvertToDouble(z, poleY.Text, «Ви ввели не число в поле
z»))
{
 PoleZ.SetFocus();
 Return;
}

 rez=( Math.Sqrt(4*4+x*x)/ Math.Sqrt(x*x+y*y))+
(Math.Sqrt(65*65+y*y)/Math.Sqrt(6,4*6,4+z*z);
poleRez.Text=rez.ToString();
```

В даній реалізації обчислення кореня з суми квадратів двох чисел виконується 4 рази. Тому доцільно створити підпрограму, яка буде обчислювати корінь з суми квадратів двох чисел і викликати цю підпрограму 4 рази.

```
        Static Double Korin( Double C1, Double C2)
             { Return Math.Sqrt(C1*C1+C2*C2);
             }
```

Тоді в процедурі обробки натиснення 2-ї кнопки замість обчислення елегантно запишемо:

```
        { rez= (Korin(4,x)/ Korin(x,y)+ Korin(65,y)/
Korin(6,4,z));
        }
```

2. Розробка і використання власних бібліотек в C#

В процесі розробки різних додатків власні універсальні функції доцільно не копіювати щоразу в нові проекти, а розмістити у власних бібліотеках і підключати та викликати при потребі, як це реалізовується, наприклад, з стандартними функціями вводу/виводу.

Бібліотеки мають розширення `.dll`. Вони самостійно не завантажуються, оскільки не містять точки входу – процедури `Main()`. В C# бібліотеки називаються *бібліотеками класів*. Для створення бібліотеки необхідно послідовно вибрати `Файл – Создать – Проект Visual C# - Библиотека классов`, перейти в папку проекту бібліотеки та ввести його ім'я. Після вдалої компіляції бібліотеки її DLL-файл буде знаходитися у вкладеній папці …`Bin\Debug`.

Кожна функція в C# має належати до якогось класу. Для того, щоб методи класу могли викликатися без створення його екземпляра-об’єкта, необхідно забезпечити статичність цих методів. З цією метою перед описом кожного такого методу вказують модифікатор `static`. Додатково, для уникнення створення об'єктів класів лише з статичними методами на початку їх опису теж можуть вказувати цей самий модифікатор.

По замовчуванню в проекті C# створюється простір імен з назвою, що співпадає з ім'ям самого проекту, хоча її й можна змінити. В цьому просторі імен доцільно функції групувати по класах з узагальнюючими назвами:

```
namespace LR7Unit
 {public static class analizText
   {public static bool rozdilZnak(char c)
     {if (c == ' ' | c == '.' | c == ',')
       return true;
      else
       return false;
     } // завершення опису функції
   } // завершення опису класу

  public static class corectInput
   {public static bool inputDouble(ref double x, string povidom)
     ...
   }
 } // завершення опису простору імен
```

Після введення тексту коду бібліотеки її необхідно запустити на виконання чи просто відкомпілювати та виправити помилки. Якщо помилки відсутні, то, незважаючи на повідомлення про неможливість запуску, буде створено її DLL-файл.

Для використання функції власної розробленої бібліотеки в **іншому** проекті необхідно:

1. Підключити в ньому файл розробленої бібліотеки. Для цього слід в панелі `Обозреватель решений` відмітити пункт `References`, обрати в його контекстному меню пункт `Добавить`, після чого натиснути кнопку `Обзор`, перейти в папку DLL-файлу бібліотеки, відмітити його і натиснути `OK`.
2. В модулі коду цього проекту відкрити простір імен розробленої бібліотеки (наприклад, при підключенні бібліотеки класів з наведеним вище кодом в модулі коду проекту вказують `using LR7Unit`);
3. В потрібних місцях коду проекту викликають розроблені методи власної бібліотеки згідно синтаксису `<назва класу>.<назва методу>` (наприклад, `analizText.rozdilZnak(S[i])`.