

## Wykład 2 Składnia języka C# (cz. 1)



## Budowa projektu



## Najprostszym przypadkiem jest aplikacja konsolowa

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

Deklaracje użytych w projekcie przestrzeni nazw (ang. *namespace*)

```
namespace ConsoleApplication2  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
        }  
    }  
}
```

Nasza przestrzeń nazw. W niej znajduje się klasa `Program` zawierająca jedną metodę o nazwie `Main`.

*Kod utworzony przez MS Visual C#*

# Pierwszy program, czyli tradycyjne „Witaj świecie”

Najprostszym przypadkiem jest aplikacja konsolowa

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

Wypisanie tekstu do  
strumienia wyjściowego  
(Out) konsoli

```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Out.WriteLine("Hello World!");
            Console.ReadKey();
        }
    }
}
```

Wczytanie znaku  
(oczekiwanie na reakcję  
użytkownika)

## Operacje we/wyjścia konsoli

---

Wyprowadzanie danych przy użyciu konsoli do strumienia wyjściowego `Out Console.Out`.

```
Console.Out.Write("Tekst");  
String s = "Ala ma kota";  
Console.Out.Write(s);
```

Wyprowadzanie łańcuchów

```
int liczba = 100;  
double pi = 3.14;  
Console.Out.Write(liczba);  
Console.Out.Write(pi);  
Console.Out.Write(liczba+" "+pi);  
Console.WriteLine("");
```

Metoda `Write` jest przeciążona i pozwala także na wyprowadzanie zmiennych liczbowych

Zwróćmy uwagę, że jeżeli w parametrze metody `Write` możemy łączyć zmienne i łańcuchy – operatorem `+`

## Operacje we/wyjścia konsoli

---

Wyprowadzanie danych przy użyciu konsoli do strumienia wyjściowego `Out Console.Out`.

```
Console.Out.Write("Tekst");  
String s = "Ala ma kota";  
Console.Out.Write(s);
```

Wyprowadzanie łańcuchów

```
int liczba = 100;  
double pi = 3.14;  
Console.Out.Write(liczba);  
Console.Out.Write(pi);  
Console.Out.Write(liczba+" "+pi);  
Console.WriteLine("");
```

Metoda `Write` jest przeciążona i pozwala także na wyprowadzanie zmiennych liczbowych

Zwróćmy uwagę, że jeżeli w parametrze metody `Write` możemy łączyć zmienne i łańcuchy – operatorem `+`

## Operacje we/wyjścia konsoli

---

Wczytywanie danych z konsoli:

Metoda `Console.WriteLine()` wczytuje linie tekstu jako łańcuch.

Wczytanie danych liczbowych wymaga zamiany łańcucha na jego wartość (parsowania)

```
int a;  
a = int.Parse(Console.ReadLine());  
Console.Out.WriteLine(a);
```

```
double p=3.6;  
p = Double.Parse(Console.ReadLine());  
Console.Out.WriteLine(p);
```



## Zmienne



Nazwa typu (słowo kluczowe, alias klasy)	Klasa z obszaru nazw System	Liczba zajmowanych bitów	Możliwe wartości (zakres)	Domyślna wartość
bool	Boolean	1 bajt = 8 bitów	true, false	false
sbyte	SByte	8 bitów ze znakiem	od -128 do 127	0
byte	Byte	8 bitów bez znaku	od 0 do 255	0
short	Int16	2 bajty = 16 bitów ze znakiem	od -32 768 do 32 767	0
int	Int32	4 bajty = 32 bity ze znakiem	od -2 147 483 648 do 2 147 483 647	0
long	Int64	8 bajtów = 64 bity ze znakiem	od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807	0L
ushort	UInt16	2 bajty = 16 bitów bez znaku	maks. 65 535	0
uint	UInt32	4 bajty = 32 bity bez znaku	maks. 4 294 967 295	0
ulong	UInt64	8 bajtów = 64 bity bez znaku	maks. 18 446 744 073 709 551 615	0L

Użyte w tej prezentacji tabelki pochodzą z książki: Visual Studio 2013. Podręcznik programowania w C# z zadaniami Autor: Matulewski Jacek



<code>char</code>	<code>Char</code>	2 bajty = 16 bitów (zob. komentarz)	Znaki Unicode od U+0000 do U+FFFF (od 0 do 65 535)	<code>'\0'</code>
<code>float</code>	<code>Single</code>	4 bajty (zapis zmiennoprzecinkowy)	Wartość może być dodatnia lub ujemna; najmniejsza bezwzględna wartość to $1,4 \cdot 10^{-45}$ , największa to ok. $3,403 \cdot 10^{38}$	<code>0.0F</code>
<code>double</code>	<code>Double</code>	8 bajtów (zapis zmiennoprzecinkowy)	Najmniejsza wartość to $4,9 \cdot 10^{-324}$ , największa to ok. $1,798 \cdot 10^{308}$	<code>0.0D</code>
<code>decimal</code>	<code>Decimal</code>	16 bajtów (większa precyzja, ale mniejszy zakres niż <code>double</code> )	Najmniejsza wartość to $10^{-28}$ , największa to $7,9 \cdot 10^{28}$	<code>0.0M</code>

Zakres większości typów „prostych” można sprawdzić za pomocą statycznych pól `MinValue` i `MaxValue`, np. `double.MaxValue`, natomiast ich rozmiar liczony w bajtach zwraca operator `sizeof`.

Wartość domyślną typu można odczytać za pomocą słowa kluczowego `default`, np. `default(int)`. W przypadku typów liczbowych jest nią zero

## Typy zmiennych

---

W języku C# wszystkie zmienne (w tym także tak zwane typy proste) są strukturami, a słowa kluczowe znane z C, C++, takie jak `int`, `double`, czy `string` są aliasami nazw tych struktur.

Stała typu `int` jest instancją struktury `System.Int32` i pozwala na dostęp do wszystkich metod tej klasy.

```
int i=10;  
string s=i.ToString();
```

Dotyczy to także stałych dosłownych (liczbowych):

```
string s = 5.ToString();
```

## Typy zmiennych

Oprócz stałych dosłownych (liczbowych) o typowej postaci, jak 1, 1.0 lub 1E0, można wykorzystywać dodatkowe stałe uzupełnione o litery określające typ stałej

### Stałe liczbowe

Zapis w kodzie C#	Typ	Opis
1	int	Liczba całkowita ze znakiem
1U	uint	Liczba całkowita bez znaku
1L	long	Liczba całkowita 64-bitowa
1F	float	Liczba
1M, 1.0M	decimal	Liczba całkowita ze znakiem o zwiększonej precyzji (dzięki zapisowi podobnemu do liczb zmiennoprzecinkowych)
1E0, 1.0E0, 1.0	double	Liczba zmiennoprzecinkowa
01		Liczba ósemkowa
0x01		Liczba szesnastkowa

## Typy zmiennych – zmienne typu var

Typ zmiennej określić można przy inicjacji przy użyciu pseudotypu `var`

```
var i = 100;  
var l = 100L;  
var s = "Wyklad";  
var f = 100.0f;  
var d = 100.0;
```

Uwaga - do zmiennej typu `var` zainicjowanej stałą typu `int` nie można już przypisać łańcucha.

Zmienne tego typu mogą być jedynie zmiennymi lokalnymi – mogą być deklarowane i inicjalizowane tylko wewnątrz metody, nie mogą być polami klas.

## Konwersja typów i rzutowanie

---

Typ zwracanej przez operator wartości zależy od typów użytych argumentów. Wybierany jest typ o większej precyzji lub większym zakresie.

```
1.0+1;    // double
1.0/2      // double, wartość 0.5
1/2        // int, wartość 0
```

Rzutowanie typów:

```
(double)1/2    //double wartość 0.5
```



## Operator



## Operatory podstawowe

Operator	Opis
<code>x.y</code>	Dostęp do pola, metody, właściwości i zdarzenia
<code>f()</code> , <code>f(x)</code>	Wywołanie metody
<code>a[n]</code>	Odwołanie do elementu tablicy lub indeksatora
<code>x++</code> , <code>x--</code>	Zwiększenie lub zmniejszenie wartości o 1 po wykonaniu innej instrukcji (np. po wykonaniu <code>int x=2; int y=x++;</code> otrzymamy <code>y=2, x=3</code> )
<code>new</code>	Tworzenie obiektu, składnia: <code>Klasa referencja=new Klasa(arg_konstr);</code>
<code>typeof</code>	Zwraca zmienną typu <code>System.Type</code> opisującą typ argumentu: <code>typeof(int).ToString()</code> równy jest <code>System.Int32</code>
<code>checked</code> , <code>unchecked</code>	Operatory kontrolujące zgłaszanie wyjątku przekroczenia zakresu dla operacji na liczbach całkowitych <code>int i = unchecked(int.MaxValue + 1);</code>





## Operatory jednoargumentowe

<code>x.y</code>	Dostęp do pola, metody, właściwości i zdarzenia
<code>f()</code> , <code>f(x)</code>	Wywołanie metody
<code>a[n]</code>	Odwołanie do elementu tablicy lub indeksatora
<code>x++</code> , <code>x--</code>	Zwiększenie lub zmniejszenie wartości o 1 po wykonaniu innej instrukcji (np. po wykonaniu <code>int x=2; int y=x++;</code> otrzymamy <code>y=2, x=3</code> )
<code>new</code>	Tworzenie obiektu, składnia: <i><code>Klasa referencja=new Klasa(arg_konstr);</code></i>
<code>typeof</code>	Zwraca zmienną typu <code>System.Type</code> opisującą typ argumentu: <code>typeof(int).ToString()</code> równy jest <code>System.Int32</code>
<code>checked</code> , <code>unchecked</code>	Operatory kontrolujące zgłaszanie wyjątku przekroczenia zakresu dla operacji na liczbach całkowitych <code>int i = unchecked(int.MaxValue + 1);</code>



## Operatory równości

`==, !=`

Wyrażenie `0.5==1/2` jest prawdziwe (wartość logiczna),  
`0.5!=1/2`, czyli `0.5!=0` jest również prawdziwe

## Operatory porównania

`<, >, <=, >=`

Porównanie wartości: `1>2` ma wartość `false`

`is`

Porównanie typów: `1 is int` ma wartość `true`

`as`

Rzutowanie równoznaczne z:

`wyrażenie is typ ? (typ)wyrażenie : (typ)null`



## Operatory dodawania i odejmowania

`+, -`

Dodawanie i odejmowanie. Typ wyniku zależy od typu argumentów, więc: `uint i=1; uint j=3; uint k=i-j;` spowoduje, że `k=4294967294`

## Operatory mnożenia

`*, /, %`

Operator dzielenia zwraca wynik zgodny z typem argumentów:  
`5/2=2, 5/2.0=2.5, 5f/2=2.5`

Operator reszty z dzielenia: `5%2=1`

## Operacje na wartościach logicznych

`&&`

Logiczne AND (`true` jedynie wtedy, gdy oba argumenty są `true`)

`||`

Logiczne OR (`true`, gdy przynajmniej jeden argument jest `true`)



## Operacje na bitach

---

$\&$	Bitowe AND (i), tzn. bit jest zapalany tylko wtedy, jeżeli oba odpowiednie bity argumentów są zapalone
------	--

$74 \& 15 = 10$  ( $01001010 \& 00001111 = 00001010$ )

---

$\wedge$	Bitowe XOR (rozłączne lub) — bit jest równy 1 tylko wtedy, gdy odpowiednie bity argumentów są różne
----------	---

$74 \wedge 15 = 69$  ( $01001010 \wedge 00001111 = 01000101$ )

---

$ $	Bitowe OR (lub) — bit jest zapalony, jeżeli przynajmniej jeden odpowiedni bit argumentów jest zapalony
-----	--

$74 | 15 = 79$  ( $01001010 | 00001111 = 01001111$ )

---

## Operatory przesunięcia bitowego

---

$\ll, \gg$	$1 \ll 1 = 2$ , bo jedynka została przesunięta z ostatniej pozycji na przedostatnią i zostało dostawione zero
------------	---

---



## Operacje na wartościach logicznych

<code>&amp;&amp;</code>	Logiczne AND (true jedynie wtedy, gdy oba argumenty są true)
<code>  </code>	Logiczne OR (true, gdy przynajmniej jeden argument jest true)

## Operatory przesunięcia bitowego

<code>?:</code>	<code>warunek?wartość_jeżeli_true:wartość_jeżeli_false</code> , gdzie <code>warunek</code> musi mieć wartość typu bool, np. <code>x&gt;y?x:y</code> zwraca większą wartość spośród <code>x</code> i <code>y</code>
-----------------	--

## Operatory przypisania

<code>=</code> oraz <code>*=</code> , <code>/=</code> , <code>%=</code> , <code>+=</code> , <code>-=</code> , <code>&lt;&lt;=</code> , <code>&gt;&gt;=</code> , <code>&amp;=</code> , <code>^=</code> , <code> =</code>	Inicjacja i zmiana wartości zmiennych, np. polecenia: <code>int x=1;</code> <code>x+=2;</code> nadadzą zmiennej <code>x</code> wartość 3
--	---



## Łańcuchy (string)



Łącuchy są implementowane w klasie `System.String` (używamy częściej aliasu `string`), tam zdefiniowane zostały metody i właściwości. Pozwalają one m.in. na porównywanie łańcuchów, analizę ich zawartości oraz modyfikacje poszczególnych znaków lub fragmentów.. Dostęp do metod klasy `String` możliwy jest zarówno wtedy, gdy dysponujemy zmienną typu `string`, jak i na rzecz stałych łańcuchowych.

```
string s = „Wyklad”;  
int dlugosc = s.Length;
```

```
int dlugosc = „Wyklad”.Length;
```



- ✓ C# używa znaków Unicode (każdy znak kodowany jest dwoma bajtami) - nie istnieje zatem problem ze znakami narodowymi.
- ✓ Przeciążony operator `+` służący do łączenia łańcuchów.
- ✓ Ciągi definiujące łańcuchy mogą zawierać sekwencje specjalne rozpoczynające się od lewego ukośnika (znaku backslash) `\` (zęsto wykorzystywane: znak końca linii `\n`, znak cudzysłowu `\,`).
- ✓ Sekwencje pozwalające definiować znaki Unicode (także spoza dostępnego na klawiaturze zestawu ASCII) zaczynają się od `\u` i numer znaku, np. `\u0048`.



# Metody klasy String

Funkcja wartość nazwa(argumenty)	Opis	Przykład użycia
<i>indeksator</i> []	Zwraca znak na wskazanej pozycji; pozycja pierwszego znaku to 0	"Helion"[0]
<i>bool</i> Equals( <i>string</i> )	Porównuje łańcuch z podanym w argumencie	"Helion".Equals("HELP")
<i>int</i> IndexOf( <i>char</i> ), <i>int</i> IndexOf( <i>String</i> )	Pierwsze położenie litery lub łańcucha; -1, gdy nie zostanie znaleziony	"Lalalalala".IndexOf('a')
<i>int</i> LastIndexOf( <i>char</i> ), <i>int</i> LastIndexOf( <i>String</i> )	Jw., ale ostatnie wystąpienie litery lub łańcucha	"Lalalalala".LastIndexOf('a')
<i>int</i> Length	Zwraca długość łańcucha, właściwość	"Helion".Length

# Metody klasy String

<code>string</code> <code>Replace(string,string),</code> <code>string Replace(char,char)</code>	Zamiana wskazanego fragmentu na inny	<code>"HELP".Replace("P","ION")</code>
<code>string Substring(int,int)</code>	Fragment łańcucha od pozycji w pierwszym argumencie o długości podanej w drugim	<code>"Wydawnictwo".Substring(5,3)</code>
<code>string Remove(int,int)</code>	Usuwa wskazany fragment	<code>"Helion".Remove(1,2)</code>
<code>string Insert(int,string)</code>	Wstawia łańcuch przed literą o podanej pozycji	<code>"Helion".Insert(2, "123")</code>
<code>string ToLower()</code> <code>string ToUpper()</code>	Zmienia wielkość wszystkich liter	<code>"Helion". ToLower()</code>
<code>string Trim()</code> oraz <code>TrimStart, TrimEnd</code>	Usuwa spacje z przodu i z tyłu łańcucha	<code>" Helion ".Trim()</code>

# Metody klasy String

<code>string PadLeft(int,char)</code> <code>string PadRight(int,char)</code>	Uzupełnia łańcuch znakiem podanym w drugim argumencie, aż do osiągnięcia długości zadanej w pierwszym argumencie	<code>"Helion".PadLeft(10,' ')</code>
<code>bool EndsWith(string),</code> <code>bool StartsWith(string)</code>	Sprawdza, czy łańcuch rozpoczyna się lub kończy podanym fragmentem	<code>"csc.exe".EndsWith("exe")</code>



string jest typem referencyjnym, lecz jego operator przypisania = zdefiniowany jest tak, że powoduje klonowanie obiektu. To dotyczy również jego metod służących do manipulacji zawartością łańcucha (nie modyfikują bieżącej instancji łańcucha, a tworzą i zwracają nowy łańcuch)

```
string s = "uzytkownikX";  
s += "domena.pl";  
s = s.Replace("X", "@");  
Console.WriteLine(s);
```

string

```
System.Text.StringBuilder sb = new System.Text.StringBuilder("uzytkownikX");  
sb.Append("domena.pl");  
sb.Replace("X", "@");  
Console.WriteLine(sb.ToString());
```

StringBuilder



## Instrukcje sterujące

## Instrukcja warunkowa if ... else

```
if (warunek) instrukcja;  
if (warunek) instrukcja; else alternatywna_instrukcja;
```

Przykład:

```
int x;  
Console.Write("x= ");  
string s = Console.ReadLine();  
x = int.Parse(s);  
  
if (x >= 0) Console.WriteLine("Liczba dodatnia");  
else Console.WriteLine("Liczba ujemna");
```

# Instrukcja wyboru switch

## Przykład:

```
Console.Write("Podaj numer dnia tygodnia: ");
string s = Console.ReadLine();
int n = int.Parse(s);
string opis;
switch (n)
{
    case 1: opis = "niedziela"; break;
    case 2: opis = "poniedziałek"; break;
    case 3: opis = "wtorek"; break;
    case 4: opis = "środa"; break;
    case 5: opis = "czwartek"; break;
    case 6: opis = "piątek"; break;
    case 7: opis = "sobota"; break;
    default: opis = "Tydzień ma tylko 7 dni"; break;
}
Console.WriteLine("Dzień tygodnia: " + n + ", " + opis);
```



```
for (inicjalizacja_indeksu ; warunek ; inkrementacja)  
    instrukcja;
```

**Przykład:**

```
int n = 100;  
string tekst = "Nauczę się programować w C#";  
for (int i=0; i<n; i++) Console.Write(i.ToString()+ " " + tekst + " ");
```





```
while (warunek) instrukcja;
```

Przykład:

```
int n = 100;  
string tekst = "Nauczę się programować w C#";  
int i=0;  
while (i<n)  
{  
    Console.Write(i.ToString()+ " " + tekst + " ");  
    i++;  
}
```

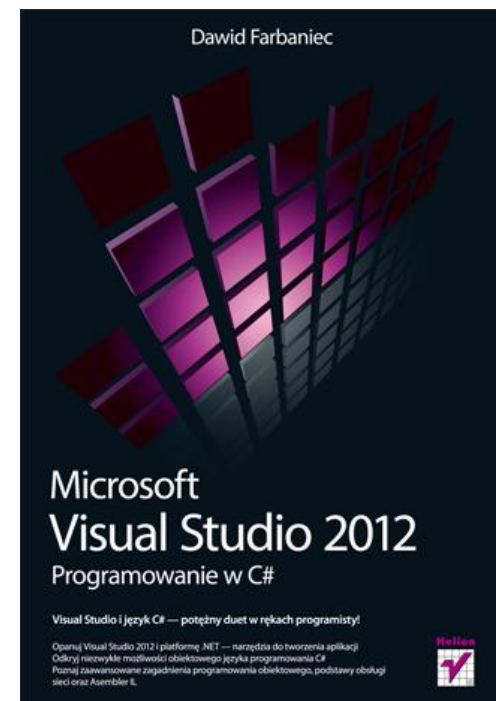
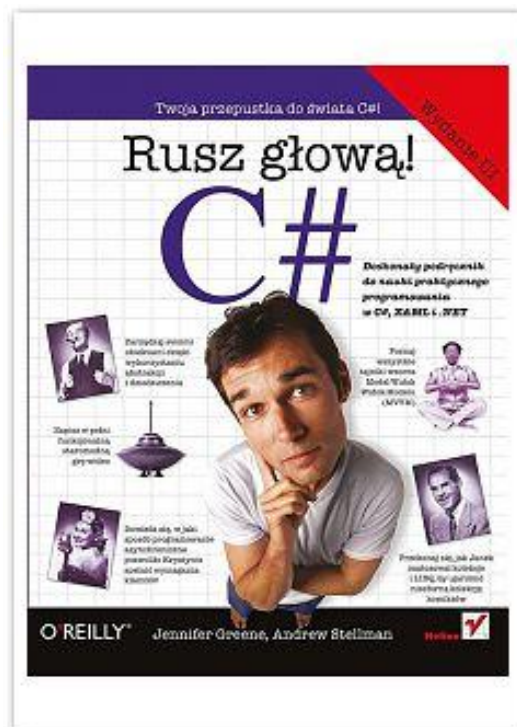
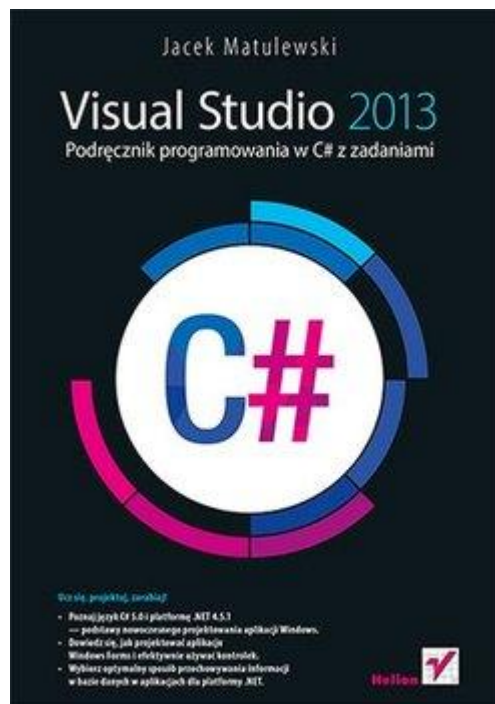
## Pętla do ... while

```
do instrukcja while (warunek);
```

Przykład:

```
int n = 100;  
string tekst = "Nauczę się programować w C#";  
int i=0;  
do  
{  
    Console.Write(i.ToString() + " " + tekst + " ");  
    i++;  
} while (i < n);  
Console.ReadKey();
```

# Literatura:



Użyte w tej prezentacji tabelki pochodzą z książki: Visual Studio 2013. Podręcznik programowania w C# z zadaniami  
Autor: Matulewski Jęcek, Helion