Wykład 1
Podstawy języka C#

dr inż. Maciej Kusy

Katedra Podstaw Elektroniki
Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska

Programowanie w języku C#

Plan wykładu

- Wybrane pozycje literaturowe, twórcy języka
- Platforma .NET
- Wprowadzenie do języka C#, idea, pisanie programów
- Typy proste (wbudowane skalarne, wartościowe)
- Znaki ucieczki
- Zmienne i stałe, automatyczna inicjalizacja zmiennych lokalnych
- Łańcuchy znaków
- Wyliczenia (typy wyliczeniowe)
- Instrukcje warunkowe i iteracyjne (pętle)
- Operatory języka C#

Literatura związana z językiem C#

- Chłosta P.: *„Aplikacje Windows Forms .Net w C#”*, Wydawnictwo naukowe PWN, 2006.
- Hejlsberg A., Torgersen M., Wiltamuth S., Golde P.: *„Język C#. Programowanie”*, Wydawnictwo Helion, 2010.
- Hilyard J., Teilhet S.: *„C#. Receptury”*, Wydawnictwo Helion, 2006.
- Kusy M.: *„Metodyki i techniki programowania. Laboratorium”*, Oficyna Wydawnicza Politechniki Rzeszowskiej, Rzeszów 2009.
- Liberty J.: *„C#. Programowanie”*, O’REILLY®, Helion, 2006.
- Lis M.: *„C#. Praktyczny kurs”*, Helion 2007.
- Matulewski J.: *„C#3.0 i .NET 3.5. Technologia LINQ”*, Helion, 2008.
- Microsoft Visual C# 2005 Express Edition: *„Projektuj sam”*, Edycja polska Microsoft Press, 2006.
- Powers L., Snell M.: *„Microsoft Visual Studio 2008 – KSIĘGA EKSPERTA”*, Helion, 2009.
- Perry S.C.: *„C# i .NET”*, Prentice Hall, Helion, 2006.
- Troelsen A.: *„Język C# 2008 i platforma .NET 3.5”*, Wydawnictwo naukowe PWN SA, 2009.
- Gamma E., Heml R., Johnson R., Vlissides J.: „*Design Patterns. Elements of Reusable Object-Oriented Software”*, Addison Wesley, 2000.

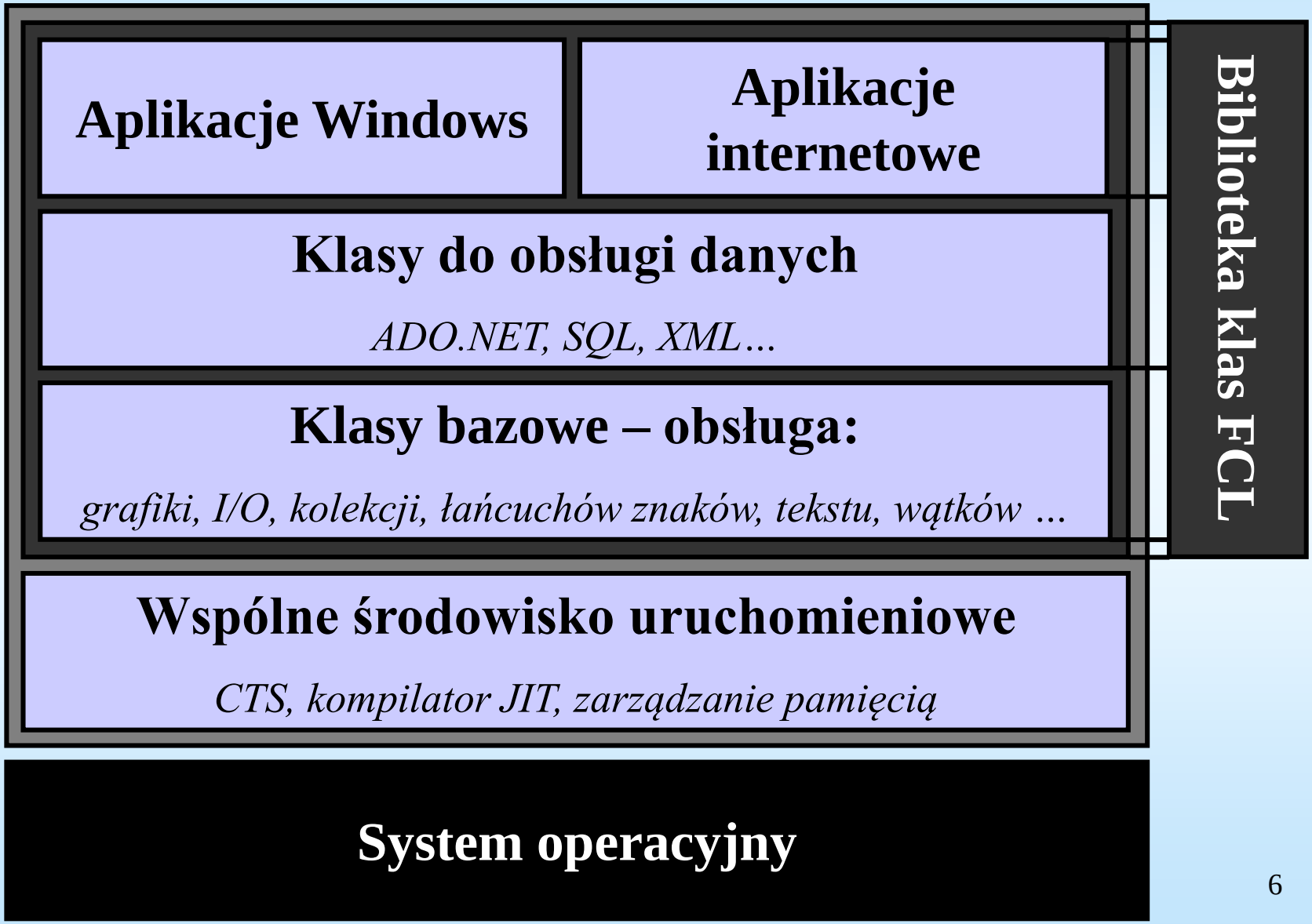
Twórcy języka

- **Anders Hejlsberg** (ur. w grudniu 1960 r.) – duński twórca oprogramowania, który uczestniczył w projektowaniu kilku popularnych języków programowania i narzędzi programistycznych:
  - kompilator **Pascal** dla mikrokomputera Nascom,
  - kompilator **Turbo Pascal** firmy **Borland** (główny architekt),
  - kompilator **Delphi** (szef zespołu produkującego następcę **TP**).
- Od 1996 r. pracuje dla firmy **Microsoft**, gdzie stworzył język **J++** i **Windows Foundation Classes**.
- Od 2000 r. wraz z **Scottem Wiltamuthem** jest głównym architektem zespołu tworzącego nowy język programowania **C#**.

Platforma .NET

- Platforma .NET: zintegrowane narzędzie (środowisko) do tworzenia i uruchamiania aplikacji.
- Funkcjonuje ponad poziomem systemu operacyjnego.
- Używa obiektowej platformy – wspólne środowisko uruchomieniowe **CLR** (***C**ommon **L**anguahe **R**untime*), na której działają języki programowania.
- Wykorzystuje liczne biblioteki klas pod wspólną nazwą **FCL** (***F**ramework **C**lass **L**ibrary*).
- Udostępnia główną klasę bazową, umożliwia dziedziczyć po klasach, przechwytywać wyjątki, korzystać z polimorfizmu, obsługiwać wspólne interfejsy, definiować delegacje dzięki wspólnemu systemowi typów **CTS** (***C**ommon **T**ype **S**ystem*).
- Dzięki **CLR** i **CTS** integruje kilka obiektowych języków programowania (m.in.: C#, Visual C++, Visual Basic, Visual J#) umożliwiając pisanie jednej aplikacji różnych języków.

Platforma .NET



Wspólne środowisko uruchomieniowe (**CLR**) – właściwości:

- Zarządza całym cyklem życia aplikacji:
  - lokalizuje i kompiluje kod,
  - wczytuje powiązane klasy,
  - zarządza wykonywaniem aplikacji,
  - zapewnia automatyczne zarządzanie pamięcią.
- Obsługuje integrację języków programowania: C#, Visual C++, Visual Basic, Visual J#.
- Odpowiada za bezproblemową współpracę kodu zapisanego w różnych językach i skompilowanych w różnych kompilatorach.

CLR: kompilacja kodu platformy .NET

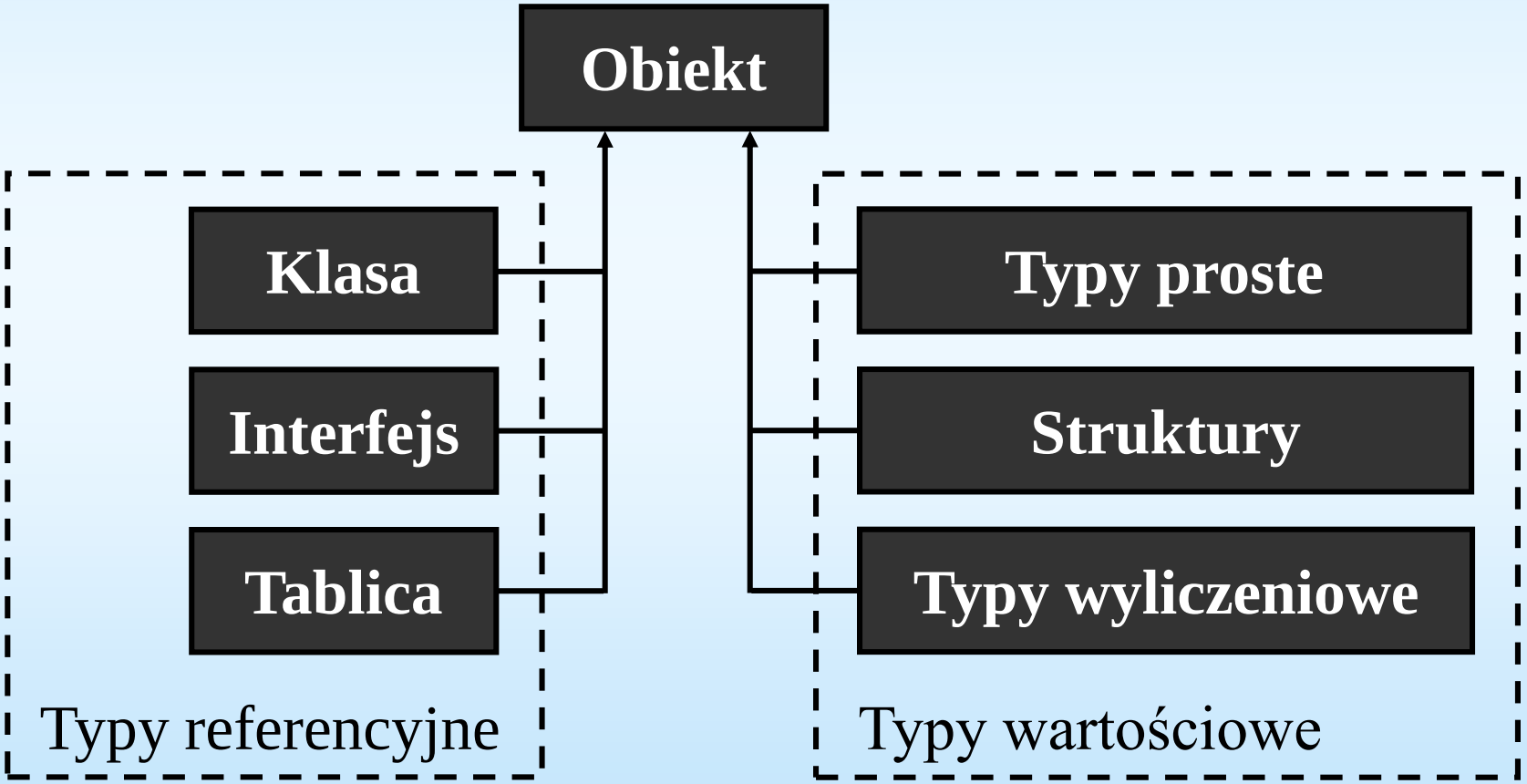
- Kompilatory spełniające wymagania **CLR** generują *kod zarządzany* dla tego środowiska.
- Kod nie jest generowany dla konkretnego procesora.
- Generowany kod – język pośredni Microsoft (**MSIL**), w skrócie **IL**, który jest niezależny od j. programowania.
- Język **IL** ma postać kodu maszynowego umieszczanego w plikach EXE lub DLL (nie są to standardowe pliki wykonywalne).
- Pliki EXE i DLL wymagają użycia kompilatora na bieżąco **JIT** (*Just In Time*) .
- **JIT** w czasie wykonywania aplikacji dokonuje konwersji języka pośredniego IL na odpowiedni kod maszynowy.

CLR: kompilacja kodu platformy .NET

- Kompilatory spełniające wymagania **CLR** generują dodatkowo **metadane** – zbiór tabel zawierający informacje o **zestawie** oraz **pakiecie** kodu, do którego należy dany kod i opis samego kodu.
- **JIT** wydobywa z metadanych informacje niezbędne do przeprowadzenia procesu kompilacji oraz weryfikacji kodu.
- Proces odpowiedzialny za odzyskiwanie pamięci (*Garbage Collection*) wykorzystuje metadane do procesu zarządzania pamięcią.

Wspólny system typów (**CTS**)

Bazowy zbiór typów danych dla każdego języka zgodnego ze specyfikacją platformy .NET:



Biblioteka klas platformy **FCL**

- Obsługa operacji wejścia i wyjścia
- Manipulacja ciągami znaków
- Zarządzanie bezpieczeństwem
- Komunikacja internetowa
- Zarządzanie wątkami
- Manipulacja tekstem
- Operacje związane z kolekcjami
- Operacje graficzne

Zasoby w ramach biblioteki **FCL** podzielono pomiędzy logiczne grupy nazwane przestrzeniami nazw (**namespace**). Istnieje ponad **4000** klas do tworzenia aplikacji na komputery stacjonarne, urządzenia przenośne, klient-serwer, usługi sieciowe. Umieszczone są one w plikach **dll**.

Wybrane przestrzenie nazw **FCL**

- **`System`** – podstawowe typy danych, klasy wyjątków i zarządzania aplikacji, biblioteka **`Math`**
- **`System.Data`** – klasy do operacji na bazach danych (**ADO.NET**)
- **`System.Drawing`** – funkcjonalność graficzna dla interfejsu programowego **GDI+** (***G**raphical **D**evice **I**nterface*)
- **`System.IO`** – operacje wejścia-wyjścia na plikach i strumieniach
- **`System.Web`** – klasy do działań w ramach aplikacji internetowych
- **`System.WindowsForms`** – klasy do budowy **GUI**
- **`System.Xml`** – typy do przetwarzania danych w formacie **XML**

Idea języka C#

- Wydajność i obiektowość języka **C++** oraz wykorzystanie takich samych technik (np. klasy abstrakcyjne, przeładowanie operatorów, struktury, typy wyliczeniowe).
- Prostota języka **Java** w tworzeniu aplikacji, ogromna liczba predefiniowanych wyjątków, wykorzystanie pojęcia interfejsu w dziedziczeniu.
- Obsługa właściwości typu i przetwarzanie w sposób iteracyjny zbioru instrukcji dla każdego elementu kolekcji danych (**`foreach`**) w ten sam sposób jak **Visual Basic**.
- Zależność od pakietu narzędziowego – platformy **.NET Framework** (C# jest w jej oparciu budowany, przez co język się rozwija; wersje: 1.0, 2.0, 3.0, 3.5, 4.0).

Gdzie pisać programy w języku C#?

- Konieczna jest instalacja pakietu .NET Framework SDK.
- Kod źródłowy umieścić w notatniku, plik zapisać z rozszerzeniem **cs** i skompilować za pomocą kompilatora uruchamianego z wiersza poleceń **csc.exe**, który jest dostępny wraz z .NET Framework SDK.
- Użyć jednego ze zgodnych z .NET narzędzi, jak Mono lub Microsoft Shared Source CLI.
- Wykorzystać zintegrowane środowisko programowania IDE (*ang. Integrated Development Entironment*) jakim jest **Visual Studio 2008**, **2010**, **2012, 2015**: mechanizm *Intellisense*, wcinanie i kolorowanie kodu, dostęp do obszernej dokumentacji pomocy, usprawnione dokowanie okien i paneli, refaktoryzacja kodu, uruchamianie programu z poziomu środowiska, automatyczne wykrywanie błędów, podglądanie zawartości zmiennych, *debugging*…

Alternatywa .NET w systemie Linux

- **Mono** – platforma programistyczna zaprojektowana w celu tworzenia różnego typu aplikacji.
- Sponsorowana przez firmę **Xamarin**.
- Jest darmową implementacją platformy .NET.
- Bazuje na CLR oraz standardach ECMA dla języka C#.
- Środowisko jest nadal rozwijane, co powoduje, że **Mono** jest wiodącym źródłem tworzenia aplikacji działających w systemie **Linux**.

Podstawy języka C#

- Wbudowane typy skalarne
- Znaki ucieczki
- Zmienne, stałe, automatyczna inicjalizacja zmiennych lokalnych
- Łańcuchy znaków
- Wyliczenia
- Instrukcje warunkowe: **`if`**, **`if else`**, **`switch`**
- Pętle: **`for`**, **`while`**, **`do while`**, **`foreach`**
- Operatory arytmetyczne, bitowe, porównania, logiczne, przypisania

Wbudowane typy proste

Typ	Rozmiar	Opis
byte	1	Bez znaku, wartości z przedziału **0** ÷ **255**
char	2	Znaki w formacie **Unicode** (komputerowy zestaw znaków mający w zamierzeniu obejmować wszystkie pisma używane na świecie)
bool	1	Wartości **true** lub **fasle**
sbyte	1	Ze znakiem, wartości z przedziału: **-128** ÷ **127**
short	2	Ze znakiem, wartości z przedziału: **-32768** ÷ **32767**
ushort	2	Bez znaku, wartości z przedziału: **0** ÷ **65535**
int	4	Liczby całkowite ze znakiem: **-2147483648** ÷ **2147483647**
uint	4	Liczby całkowite bez znaku: **0** ÷ **4294967295**
float	4	Liczby zmiennoprzecinkowe: $-3.4 \cdot 10^{38} \div 3.4 \cdot 10^{38}$ (7 cyfr): przyrostek **f**, **F**
double	8	Liczby zmiennoprzecinkowe podwójnej precyzji: $\pm 5.0 \cdot 10^{-324} \div \pm 1.7 \cdot 10^{308}$ (**16** cyfr)
decimal	16	Liczby zmiennoprzecinkowe: $-7.9 \cdot 10^{-28} \div 7.9 \cdot 10^{28}$ (**28-29** cyfr): przyrostek **m**, **M**
long	8	Liczby całkowite ze znakiem: **-9223372036854775808÷9223372036854775807**
ulong	8	Liczby całkowite bez znaku, wartości: od **0** do **18446744073709551615**
Object	-	Ostateczny typ bazowy dla wszystkich typów
?	-	Typ adresu zerowego (nullable), np: **int?**, **bool?**, **double?**

Znaki ucieczki

Znak	Znaczenie
`\'`	Apostrof
`\"`	Cudzysłów
`\\`	Odwrócony ukośnik, można ominąć poprzez `@`
`\0`	Znak **`null`**
`\a`	Znak alarmu
`\b`	Znak cofania
`\f`	Znak przesunięcia strony
`\n`	Nowy wiersz
`\r`	Powrót karetki
`\t`	Tabulacja pozioma
`\v`	Tabulacja pionowa

Pojęcie zmiennej i stałej

Zmienna – element (instancja) o określonym typie oraz nazwie w obszarze pamięci. Zmienne, w momencie definicji podlegają przypisaniu (inicjalizacji) przy użyciu operatora przypisania (=). Przechowują wartość, która może ulegać zmianom, np.:

```
int temperaturaPowietrza = 22;
temperaturaPowietrza = 29;
```

Stała – instancja (wartość symboliczna) danych o określonym typie oraz nazwie, która musi zostać zainicjalizowana w miejscu deklaracji i nie może ulec jakiejkolwiek modyfikacji:

```
const double pi = 3.141592654;
pi = 3.14; //BŁĄD
```

Automatyczna inicjalizacja zmiennych

W wersji **C# 3.0** wprowadzono możliwość deklaracji zmiennych typów prostych przy użyciu słowa kluczowego **`var`**.

Typ tych zmiennych ustalany jest automatycznie przez kompilator na podstawie wartości użytej do inicjalizacji:

```
var i = 3;
var j = 3.333;
var k = "wykład";
```

Inicjalizacja musi nastąpić w tej samej linii kodu co deklaracja zmiennej.

Zmienne te muszą być zmiennymi lokalnymi (wewnątrz metody).

Łańcuchy znaków

Łańcuchy znaków w języku C# przechowywane przez obiekty typu **`string`** reprezentują sekwencję zera lub więcej znaków formatu Unicode.

```
string znaki = "Programowanie w C#";

System.Console.WriteLine(znaki);
```

Metoda wypisująca argument z przejściem do nowej linii.

Metoda wczytująca całą linię znaków wpisanych z klawiatury.

```
znaki = System.Console.ReadLine();

System.Console.WriteLine(znaki);
```

Wyliczenia (typy wyliczeniowe)

Wyliczenia – odrębny typ prosty, składający się ze zbioru stałych symbolicznych o określonych nazwach (*lista wyliczeniowa*) i wartościach całkowitych:

```
enum DniPracy : short
{
    Pon = 1, Wto = 2, Sro, Czw, Pia = 5
}
```

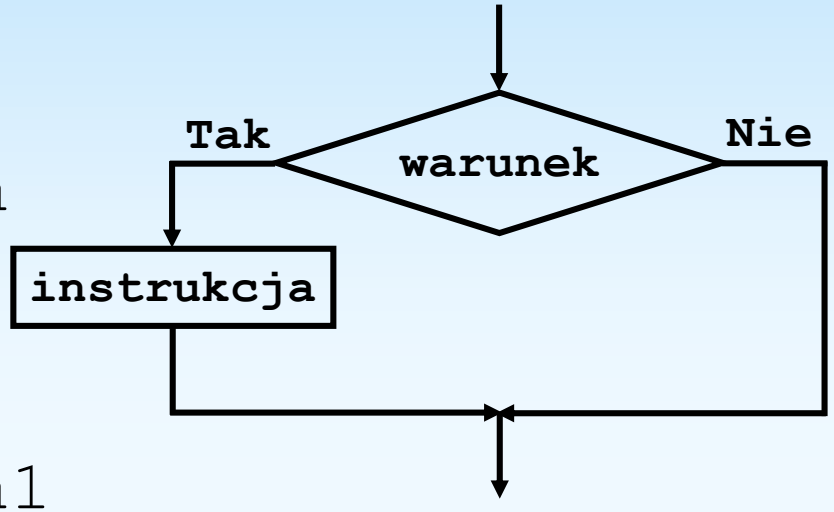
domyślnie **int**

Stworzenie zmiennej typu wyliczeniowego:

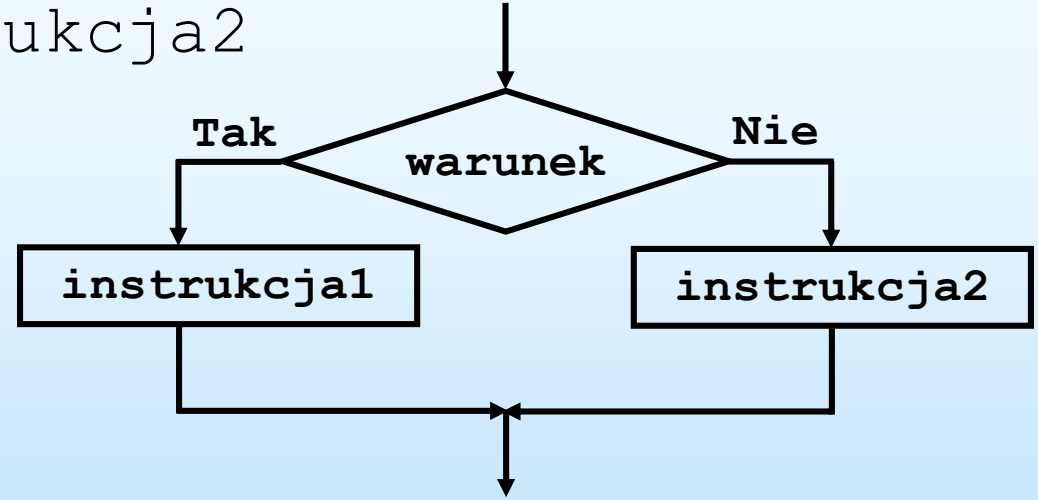
```
DniPracy dzien = DniPracy.Wto;
dzien = DniPracy.Sro;

System.Console.WriteLine((int)dzien); // 3
```

Instrukcje warunkowe **if**, **if else**, **if else if**

```
if (warunek) instrukcja
```




```
if (warunek) instrukcja1
else         instrukcja2
```



if else if

Instrukcja warunkowa **switch**

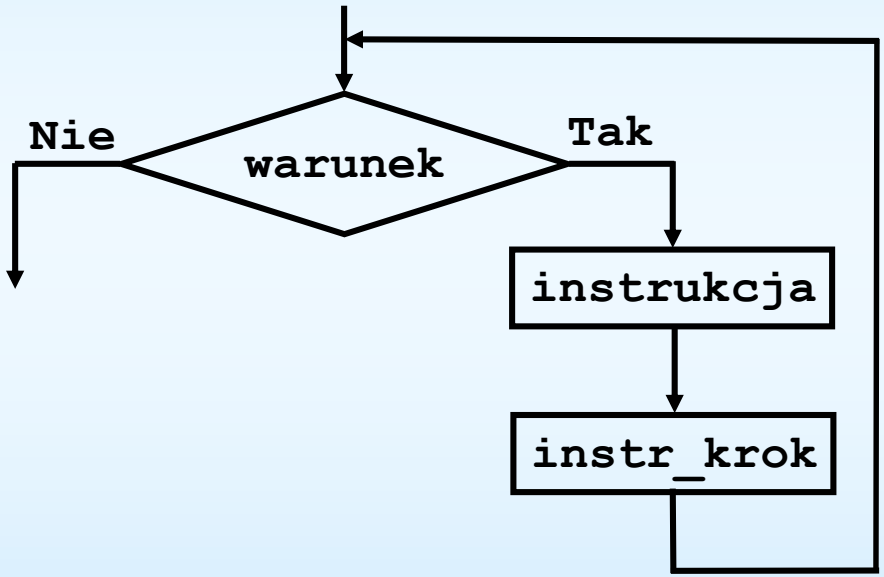
```
switch(wyrażenie)
{
  case wyrażenie stałe:
     instrukcje;
     break/goto/return;
  case wyrażenie stałe:
     instrukcje;
     break/goto/return;
  default:
     instrukcje;
     break/goto/return;
}
```

Uwaga:

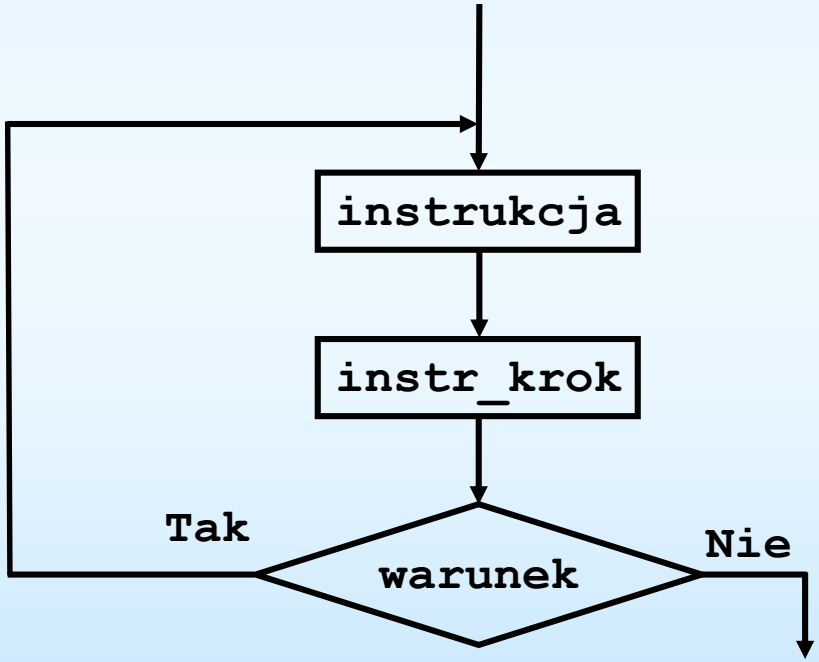
- Wyrażenie stałe może być liczbą całkowitą, znakiem, wartością typu wyliczeniowego lub łańcuchem znaków.
- Każdy blok **case** musi kończyć się instrukcją przenoszącą sterowanie.
- Możliwe jest łączenie przypadków.

Pętle **while** i **do while**

```
while (warunek)
      instrukcja
```

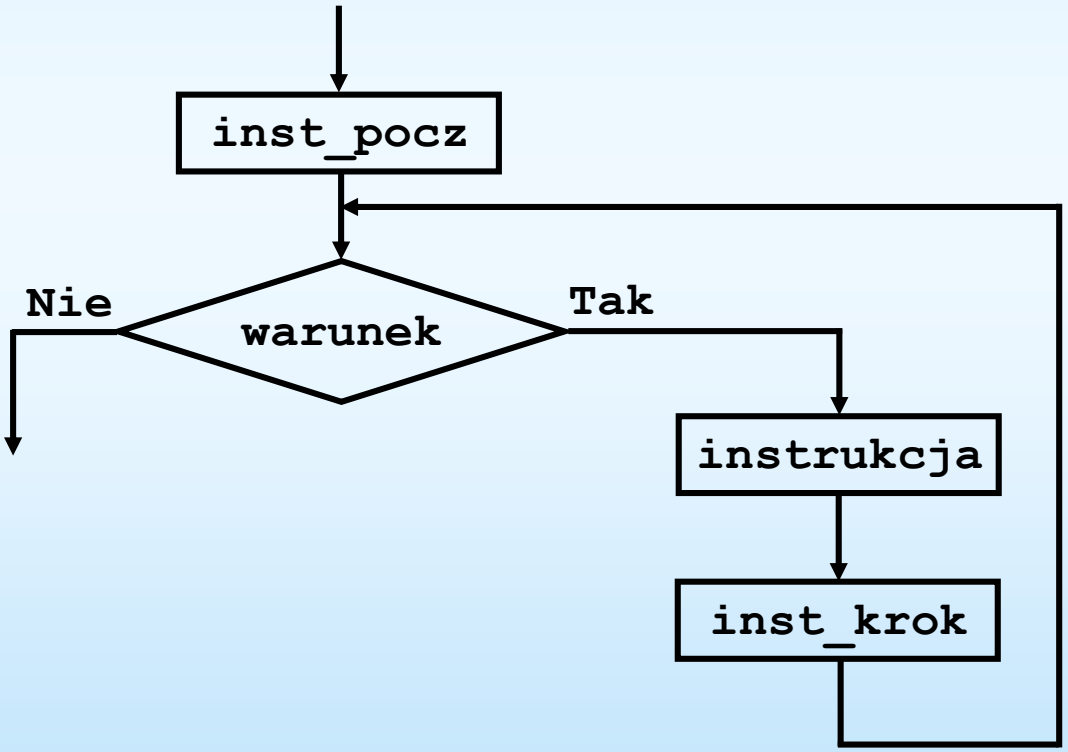



```
do
   instrukcja
while (warunek);
```



Pętla **for**

```
for (inst_pocz; warunek; inst_kroku)
     instrukcja
```



Pętla **foreach**

Instrukcja iteracyjna – służy do przetwarzania w sposób iteracyjny zbioru instrukcji dla każdego elementu kolekcji danych.

```
foreach (typ iterator in tablica)
     instrukcja
```

```
int [] tab = {1,2,3,4,5};
foreach (int i in tab)
{
     System.Console.WriteLine(i);
}
```

Przy przesyłaniu tablicy do metody nie jest konieczne podawanie jej długości.

Operatory języka C#

Kategoria	**Operatory**
arytmetyczne	`+ - * / % ++ --`
bitowe	`~ >> << & | ^`
porównania	`== != > < >= <=`
logiczne	`&& || ! ??`
przypisania	`= += -= *= /= %= >>= <<= |= &= ^=`

Podstawowe operatory języka C# (wybrane):

`is as new typeof sizeof (x) x.y f(x) a[x]`