Podstawy programowania

mgr inż. Katarzyna Dadek

2024

Lista 3
Podstawy C#

Platforma .NET (zwłaszcza .NET Core) to wieloplatformowe środowisko uruchomieniowe opracowane przez Microsoft, które umożliwia tworzenie aplikacji na różne urządzenia (Windows, Linux, macOS). C# jest językiem programowania wysokiego poziomu zaprojektowanym na platformę .NET, który integruje różne usługi systemowe, takie jak zarządzanie pamięcią, bezpieczeństwo i optymalizacja kodu.

3.1 Podstawowe elementy

3.1.1 Typy danych

C# jest językiem silnie typowanym, co oznacza, że każda zmienna musi mieć zdefiniowany typ. Podstawowe typy danych dzielą się na dwa główne rodzaje:

- **typy wartościowe** - przechowujące rzeczywiste dane, np. int, float, double, char, bool, struct;
- **typy referencyjne** - przechowujące adresy pod którymi znajdują się dane, np. string, class, object, array.

3.1.2 Zmienne

Zmienne są nazwanymi miejscami w pamięci, które przechowują dane. Zmienna musi być zadeklarowana przed użyciem, np.:

```
1  int liczba;
2  liczba = 10;
```

Typ zmiennej determinuje, jakie operacje można na niej wykonywać oraz jak dużo pamięci potrzebuje.

3.1.3 Operatory

C# posiada zestaw operatorów pozwalających na wykonywanie działań na zmiennych. Można je podzielić na:

- **arytmetyczne**: `+`, `-`, `*`, `/`, `%` (modulo);

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  int sum = a + b;
  int diff = a - b;
  int prod = a * b;
  int quotient = a / b;
  int mod = a % b;
 }
}
```

- **porównania**: `==`, `!=`, `>`, `<`, `>=`, `<=`;

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  bool isEqual = (a == b);
  bool isNotEqual = (a != b);
  bool isGreater = (a > b);
  bool isLesserOrEqual = (a <= b);
 }
}
```

- **logiczne**: `&&` (AND), `||` (OR), `!` (NOT);

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  bool resultAnd = (a > 0 && b > 0); // AND
  bool resultOr = (a > 0 || b > 0);  // OR
  bool notResult = !(a > b);         // NOT
 }
}
```

- **bitowe**: &, |, ^, ~, «, »;

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  int bitAnd = a & b;
  int bitOr = a | b;
  int bitXor = a ^ b;
  int bitNot = ~a;
  int leftShift = a << 1;
  int rightShift = b >> 1;
 }
}
```

- **przypisania**: =, +=, -=, *=, /=, %=;

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  a += b;
  b -= 2;
 }
}
```

- **inne**: ?: (operator warunkowy), is, as.

```
class Program {
 static void Main() {
  int a = 10;
  int b = 5;

  // Conditional operator
  string message = (a > b) ? "a > b" : "a <= b";

  // Type operators
  bool isInt = a is int;
 }
}
```

3.2 Instrukcje

Instrukcje sterujące przepływem programu umożliwiają podejmowanie decyzji, powtarzanie operacji i kontrolowanie zachowania aplikacji. Najważniejsze to:

- **if** - instrukcja warunkowa;

```
class Program {
 static void Main() {
  int x = 10;

  if (x > 5) {
   Console.WriteLine("x > 5");
  } else {
   Console.WriteLine("x <= 5");
  }
 }
}
```

- **switch** - instrukcja wyboru;

```
class Program {
 static void Main() {
  int x = 10;

  switch (x) {
   case 1:
   Console.WriteLine("x = 1");
   break;
   case 10:
   Console.WriteLine("x = 10");
   break;
   default:
   Console.WriteLine("x has other value");
   break;
  }
 }
}
```

- **for, while, do-while** - pętle;

```
for (int i = 0; i < 10; i++) {
 Console.WriteLine("Value of i: " + i);
}
```

```
while (x > 0) {
 Console.WriteLine("x wynosi: " + x);
 x--;
}
```

```
do {
 Console.WriteLine("x: " + x);
 x++;
} while (x < 5);
```

3.2.1 Zarządzanie pamięcią operacyjną

Zarządzanie pamięcią operacyjną w .NET odbywa się automatycznie przez mechanizm nazywany **Garbage Collector** (GC). GC śledzi obiekty utworzone w trakcie działania aplikacji i usuwa te, które nie są już używane, aby zwolnić pamięć. Użytkownicy nie muszą ręcznie zarządzać pamięcią (tak jak w C++), co zmniejsza ryzyko wycieków pamięci.

3.2.2 Garbage Collector

GC śledzi obiekty w pamięci **heap** (sterta) i usuwa te, które nie mają żadnych referencji, tzn. nie są już używane przez program. Proces ten odbywa się w kilku etapach:

1. Znajdowanie obiektów osieroconych - GC identyfikuje obiekty, które nie są już dostępne dla żadnego żyjącego kodu (brak referencji).
2. Kompaktowanie pamięci - po usunięciu zbędnych obiektów, GC może przenieść pozostałe obiekty w nowe miejsca w pamięci, aby wyeliminować fragmentację.
3. Przywracanie wskaźników - wskaźniki do obiektów, które zostały przemieszczone, są aktualizowane.

GC wykonuje swoją pracę w tle, jednak może zostać uruchomiony automatycznie, gdy:

- brakuje wolnej pamięci,
- system uzna, że warto zwolnić zasoby,
- programista ręcznie wymusi kolekcję za pomocą metody GC.Collect().

3.2.3 Pokolenia

Pokolenia (ang. *generations*) to sposób organizacji obiektów w stercie zarządzanej przez GC. Obiekty są podzielone na trzy *pokolenia*, co optymalizuje proces zarządzania pamięcią.

- Pokolenie 0: obiekty nowe, krótkotrwałe (np. lokalne zmienne), GC najczęściej sprawdza i czyści to pokolenie, ponieważ wiele obiektów staje się szybko niepotrzebnych.
- Pokolenie 1: obiekty, które przetrwały pierwszą kolekcję. Wykorzystywane jako bufor między pokoleniem 0 a 2.
- Pokolenie 2: obiekty długowieczne (np. dane aplikacji, cache), które przetrwały wiele rund kolekcji. Obiekty w tym pokoleniu są rzadziej sprawdzane przez GC, ponieważ zakłada się, że będą dłużej potrzebne.

Dzięki podziałowi na pokolenia, GC może działać bardziej efektywnie, koncentrując się na obiektach krótkotrwałych (pokolenie 0) i rzadziej sprawdzając te długowieczne (pokolenie 2).

3.2.4 Destruktory

Są to specjalne metody w C#, które są automatycznie wywoływane przez GC, zanim obiekt zostanie usunięty z pamięci. Ich głównym celem jest zwolnienie zasobów niezarządzanych (np. uchwyty do plików, zasoby systemowe) lub wykonanie innych czynności sprzątających przed zniszczeniem obiektu. Destruktor jest definiowany za pomocą "∼"(tzw. ∼ClassName):

```
1  class ExampleClass {
2   ~ExampleClass() {
3    // to do
4   }
5  }
```

Wady użycia destruktorów:

- opóźniają zwolnienie pamięci, ponieważ obiekt nie jest usuwany w pierwszym przebiegu GC – jest usuwany dopiero po dodatkowej rundzie kolekcji;
- mogą wprowadzać nieprzewidywalność w czasie wykonania, ponieważ nie wiadomo, kiedy dokładnie GC go uruchomi;

Z tego powodu lepiej unikać destruktorów, jeśli jest możliwość użycia bardziej kontrolowanych mechanizmów, np. implementacji interfejsu *IDisposable*.

3.2.5 Zasoby niezarządzane

Zasoby niezarządzane (ang. *unmanaged resources*) to zasoby systemowe, które nie są automatycznie zarządzane przez .NET i GC. Przykłady to:

- uchwyty do plików;
- połączenia sieciowe;
- połączenia z bazami danych;
- wskaźniki do pamięci systemowej.

Ponieważ GC nie jest odpowiedzialny za zarządzanie tymi zasobami, programista musi ręcznie je zwalniać. Najczęściej odbywa się to poprzez implementację interfejsu **IDisposable** i metody **Dispose()**, która zwalnia zasoby niezarządzane, gdy obiekt nie jest już potrzebny.

Linki

Poniżej podano linki zawierające dokładniejszy opis powyższych zagadnień wraz z przykładami oraz dodatkowe elementy przydatne w rozwiązaniu poniższych zadań.

- `https://www.w3schools.com/cs/cs_type_casting.php`
- `https://www.w3schools.com/cs/cs_user_input.php`
- `https://www.w3schools.com/cs/cs_operators.php`
- `https://www.w3schools.com/cs/cs_math.php`
- `https://www.w3schools.com/cs/cs_strings.php`
- `https://www.w3schools.com/cs/cs_booleans.php`
- `https://www.w3schools.com/cs/cs_conditions.php`
- `https://www.w3schools.com/cs/cs_switch.php`
- `https://www.w3schools.com/cs/cs_while_loop.php`
- `https://www.w3schools.com/cs/cs_for_loop.php`
- `https://www.w3schools.com/cs/cs_break.php`

3.3 Zadania

Ważna informacja

Każde zadanie wymagające stworzenia kodu powinno być osobnym projektem w solucji. W poniższych przykładach widoków *konsoli* kolorem ***cyan*** zaznaczono dane wprowadzane przez użytkownika.

Z3.1. Zapoznaj się z materiałami powyżej - wstępem teoretycznym oraz linkami.

Z3.2. Napisz program, który zapyta użytkownika o jego imię, pozwoli na jego wpisanie i następnie je wyświetli na ekranie. Przykład:

```
> Proszę podać imię: Kasia
> Witaj Kasia!
```

Z3.3. Wczytaj dwie liczby z konsoli a, b oraz sprawdź która z nich jest większa (mogą być równe). Przykład:

```
> Proszę podać liczbę A: 100
> Proszę podać liczbę B: 90
> Liczba 100 jest większa od 90.
```

Z3.4. Utwórz program, który na podstawie wartości zmiennej wiek wczytanej od użytkownika decyduje, czy użytkownik jest pełnoletni (18+) lub nie, stosując instrukcję if.

Z3.5. Napisz program, który prosi użytkownika o podanie liczby godzin i zamienia je na minuty i sekundy.

Z3.6. Napisz program, który przeliczy wartość z PLN na USD oraz EURO. Wartość wraz z walutą powinna być wczytywana z konsoli. Użyj instrukcji switch ... case. Przykład:

```
> Proszę podać wartość: 100
> Proszę podać walutę wejściową [PLN, USD, EUR]: USD
> Proszę podać walutę docelową [PLN, USD, EUR]: PLN
> Wynik: 100USD = 450PLN
```

Z3.7. Utwórz program, który pobiera liczbę od użytkownika i sprawdza, czy jest ona dodatnia, ujemna, czy równa zero.

Z3.8. Napisz program, który prosi użytkownika o podanie dwóch liczb całkowitych, wykona operacje arytmetyczne (+, -, *, /, %) na tych liczbach i wyświetli wyniki. Przykład:

```
> A: 50
> B: 10
> A + B = 50 + 10 = 60
> A - B = 50 - 10 = 40
> A * B = 50 * 10 = 500
> A / B = 50 / 10 = 5
> A % B = 50 % 10 = 0
```

Z3.9. Napisz prosty kalkulator pozwalający na proste operacje na wartościach typu double. Program ma umożliwiać operacje takie jak suma, różnica, iloczyn oraz iloraz. Operacja powinna być wybierana z klawiatury - wpisanie odpowiedniego znaku (+, -, /, *) definiuje jaką operacje ma wykonać program. Przykład:

```
> Proszę podać wartość A: 10
> Proszę podać wartość B: 5
> Proszę podać operator [+, -, *, /]: +
> Wynik: 10 + 5 = 15
```

Z3.10. Napisz program, który wypisuje liczby od 1 do 100, ale pomija liczby podzielne przez 3, używając pętli for.

Z3.11. Napisz program, który sumuje liczby od 1 do 100 i wyświetla wynik.

Z3.12. Napisz program, który oblicza pole trójkąta na podstawie podanej przez użytkownika podstawy i wysokości.

Z3.13. Napisz program wczytujący liczbę całkowitą dodatnią n i wypisujący czy jest ona mniejsza lub równa 10, większa od 10 ale mniejsza lub równa 100, większa od 100 ale mniejsza lub równa 1000, większa od 1000. Użyj instrukcji switch.

Z3.14. Napisz program, który pobiera liczbę i wyświetla jej tabliczkę mnożenia od 1 do 10.

Z3.15. Utwórz program, który liczy sumę cyfr liczby podanej przez użytkownika.

Z3.16. Napisz program, który wypisuje wszystkie liczby parzyste od 1 do 100.

Z3.17. Utwórz program, który wypisuje liczby od 1 do 50, ale zamienia każdą liczbę podzielną przez 5 na "Hello!".

Z3.18. Narysuj w konsoli wypełniony prostokąt z gwiazdek o szerokości 5 i wysokości 3.

Z3.18. Narysuj w konsoli wypełniony prostokąt z gwiazdek o szerokości N i wysokości M. Wartości N i M mają być podane przez użytkownika i mogą być w zakresie jedynie od 1 do 100.

Z3.19. Napisz program, który sprawdza, czy liczba podana przez użytkownika jest liczbą pierwszą.

Z3.20. Napisz program, który prosi użytkownika o podanie długości trzech boków trójkąta i sprawdza, czy mogą one tworzyć trójkąt (warunek trójkąta).

Z3.21. Za pomocą pętli for, while oraz do...while oblicz sumę liczb od 1 do 10.

Z3.22. Napisz program, który za pomocą instrukcji for dla danych wartości x zmieniających się w przedziale 0-10 obliczy wartość funkcji y = 3x.

Z3.23. Napisz program używając pętli while który policzy silnię liczby podanej przez użytkownika.

Z3.24. Napisz program, który za pomocą instrukcji do...while sumuje liczby nieparzyste całkowite z przedziału podanego przez użytkownika.

Z3.25. Wiedząc, że $1233 = 12^2 + 33^2$ (liczba po lewej stronie równania podzielona na dwie liczby składowe, tu 12 i 33 wynosi tyle samo co te liczby podniesione do potęgi i zsumowane), napisz program, który znajdzie wszystkie liczby z przedziału od 1000 do 9999 spełniające tą zależność, wyświetli je i poda ich liczbę.

Z3.26. Napisz prostą grę konsolową pozwalająca na rozegranie partii gry *Kółko i krzyżyk* na 9 kratek (3x3).

Program na starcie powinien umożliwiać podanie nazw graczy. Gra ma polegać na następujących sekwencjach:

- Wyświetlenie widoku tabelki;
- Zapytanie użytkownika A o koordynaty (np. może on wpisać wartość 22 co oznacza środek planszy) oraz ich sprawdzenie (czy może je podać);
- Wyświetlenie zaktualizowanego widoku tabelki;
- Sprawdzenie wygranej;
- Zapytanie użytkownika B o koordynaty;

– Sprawdzenie wygranej;

– W przypadku wygranej wyświetlenie komunikatu i nazwy zwycięzcy;

Przykład działania:

```
> Nazwa gracza X Dante
> Nazwa gracza O: Vergil
> Losowanie kto rozpoczyna...
> Rozpoczyna gracz: Dante (X)
> _ | _ | _
> _ | _ | _
> _ | _ | _
> Dante (X) podaj pozycję: 22
> _ | _ | _
> _ | X | _
> _ | _ | _
> Vergil (O) podaj pozycję: 11
> O | _ | _
> _ | X | _
> _ | _ | _
...
> Dante (X) podaj pozycję: 13
> O | X | O
> _ | X | _
> X | X | O
> Wygrywa gracz Dante (X)!
```

Zacznij od przygotowania i rozrysowania schematu działania aplikacji!