giganci programowania

Najczęstsze komendy

Wstęp do programowanie w C#

Witaj w materiałach pomocniczych przygotowanych specjalnie dla Ciebie. Znajdziesz tutaj najczęstsze kody i komendy potrzebne przy programowaniu. Zapisz, wydrukuj i korzystaj! Powodzenia!

Najczęściej używane komendy:

1. Zmienne

Wartości liczbowe i znakowe w programie przechowywane są w zmiennych. Aby pobrać wartość zmiennej należy odwołać się do jej nazwy. Każda zmienna wymaga zainicjalizowania, czyli należy ją zadeklarować i przypisać wartość początkową.

Deklaracja zmiennej:
typ zmiennej nazwa zmiennej = wartość zmiennej;

W języku C# występuje kilka typów zmiennych takich jak: liczby całkowite (**int**, **long**), liczby zmiennoprzecinkowe (**float**, **double**, **decimal**), zmienne znakowe (**char**, **string**), zmienne logiczne (**bool**).

```
// stworzenie zmiennej typu int o nazwie a. Każdą instrukcję w C# kończymy średnikiem: ;
int a;
// stworzenie zmiennej oraz przypisanie do niej wartości 5;
int b = 5;
// do raz utworzonej zmiennej przypisywać możemy wartości w dowolnym momencie.
// Typ zmiennej podajemy tylko w momencie jej definiowania.
int c;
c = 5;  // potem odwołujemy się już tylko po jej nazwie.
// int - typ na liczby całkowite
int liczbyCalkowite = 4;
// double - typ na liczby niecałkowite (podwójnej precyzji)
double liczbyZmiennoprzecinkowe = 4.56;
// float - liczby niecałkowite, podobnie jak double ale mniejszej precyzji
// (Na końcu liczby trzeba dopisać literkę 'f' lub 'F')
float liczbyZmiennoPrzecinkowe2 = 3.561f;
```

```
// decimal - typ wykorzystywany do operacja na pieniądzach,
// na końcu dopisujemy literkę 'm' lub 'M' (od ang: Money)
decimal liczbyStaloPrzecinkowe = 3.4000m;
// typ na tekst (łańcuchy znaków), teksty zawsze zapisujemy w cudzysłów
string teksty = "Ala ma kota";
// typ na pojedynczy znak, zapisujemy w pojedynczych apostrofach ''
char znak = 'A';
// typ logiczny przechowuje dwie wartości false lub true
// - informuje czy coś jest prawdą lub nie
bool logiczny = false;
```

Czasami zachodzi potrzeba, aby raz nadana wartość w zadeklarowanej zmiennej nie mogła zostać zmieniona. W tym celu można przekształcić ją w stałą dodając przed typem słowo kluczowe **const.**

```
const double pi = 3.14159; // deklarując stałą trzeba ją od razu zainicjować (nadać jej wartość)
pi = 5.1; // to przypisanie spowoduje błąd
```

2. Operatory matematyczne

Na zmiennych można wykonywać operacje matematyczne tak jak na liczbach.

```
int liczbaA = 10;
int liczbaB = 15;

int suma = liczbaA + liczbaB;
int różnica = liczbaA - liczbaB;
int dzielenieCałkowite = liczbaA / liczbaB;
int mnożenie = liczbaA * liczbaB;
int resztaZDzielenia = liczbaA % liczbaB;
```

3. Operatory logiczne

Na zmiennych wykonywać można operacje logiczne. Wynikiem operacji logicznej jest zawsze wartość logiczna (prawda lub fałsz), czyli wartość typu **bool.** Wyrażeń logicznych używamy do sprawdzenia czy jakieś zdanie jest prawdziwe, czy spełnia jakieś warunki.

Dostępne operatory logiczne:

> większę

>= większę lub równe

< mniejsze

<= mniejsze lub równe

== równe (nie mylić ze znakiem = -jest to operator przypisania)

!= różne (przeciwieństwo ==)

&& jest to matematyczna koniunkcja (daje prawdę, gdy warunek po lewej i prawej stronie jest spełniony), określana inaczej jako logiczne **'and'**, **'i'**;

|| matematyczna alternatywa (daje prawdę, gdy chociaż jeden warunek jest prawdziwy) - określana jako logiczne 'or' 'lub'

! jest to negacja

```
int a = 10;
int b = 5;
// wynikiem zastosowania operatorów jest zawsze prawda lub fałsz - czyli wartości typu bool
bool wynik;

// jeżeli a jest większę od b to wynikiem będzie wartość true
wynik = a > b;
// jeżeli a jest większe lub równe od b to wynikiem będzie wartość true
wynik = a >= b;
// jeżeli a jest mniejsze od b to wynikiem będzie wartość true
wynik = a < b;
// jeżeli a jest mniejsze lub równe od b to wynikiem będzie wartość true
wynik = a <= b;
// jeżeli a jest równe od b to wynikiem będzie wartość true
wynik = a == b;
// jeżeli a jest różne od b to wynikiem będzie wartość true
wynik = a != b;
// operator ! - neguje wynik
wynik = !true;
// sprawdzamy dwa warunki i oba muszą być prawdziwe, żeby w wyniku dostać prawdę
wynik = (a == b) && (a > 5);
```

```
// sprawdzamy warunki i wystarczy, że pierwszy sprawdzony warunek będzie prawdą,
// żeby dostać w wyniku prawdę
wynik = (a == b) || (a > 5);
```

4. Instrukcje warunkowe

Instrukcji warunkowej używamy w celu sterowania przebiegiem programu. Jeżeli po sprawdzeniu warunku chcemy wykonać więcej niż jedną instrukcję należy umieścić je pomiędzy klamrami { }.

```
int liczba = 1;
// wynikiem działania w warunku musi być prawda lub fałsz
if (liczba > 0)
{
    // jeżeli ten warunek będzie spełniony ten kod
}
// instrukcji else if może być ile chcemy
else if (liczba < 0)
{
   // jeżeli ten warunek będzie spełniony ten kod
}
else
{
   // fragment z else wykona się gdy nie spełniony będzie żaden z warunków
}
```

5. Pętle

Pętle w programowaniu wykorzystujemy do wykonania pewnych fragmentów kodu wielokrotnie.
W C# występują cztery rodzaje pętli: **for**, **while**, **do ... while** oraz **foreach.**
Pętla **for** – wykorzystujemy ją w przypadku, gdy chcemy powtórzyć pewien fragment kodu określoną ilość razy.

for (inicjalizacja zmiennej; warunek logiczny; zaktualizowanie zmiennej)
{

Kod programu;

}

```
// zasada działania:
// na początku pętli tworzona jest zmienna i,
// która będzie zliczać ilość wykonań pętli (int i = 0)
// następnie sprawdzany jest warunek (i < 10), jeżeli warunek jest spełniony
// to następuje wykonanie kodu zawartego w ciele pętli { },
// następnie zmienna i jest zwiększana o 1 (i++)
// znowu sprawdzany jest warunek... i tak dalej, dopóki warunek jest spełniony
for (int i = 0; i < 10; i++)
{
   // kod programu, który ma się powtarzać
}
```

Pętla **while** – wykorzystujemy ją w sytuacji, gdzie ilość powtórzeń pętli zależeć będzie od jakiegoś logicznego warunku.

```
// pętla while wymaga podobnie jak instrukcja warunkowa podania warunku.
// pętla będzie wykonywana póki warunek ma wartość true
// może się okazać, że warunek na samym początku będzie mieć wartość false
// i wtedy pętla nie wykona się ani razu
bool warunek = true;
int iterator = 1;
while (warunek)
{
   iterator++;
   // Jeżeli iterator osiągnie wartość równą lub większą 5 to ustawiamy warunek na false,
   // przez co nie nastąpi już kolejne wykonanie pętli
   if (iterator >= 5)
   {
     warunek = false;
   }
}
```

Pętla **do**...**while** – wykorzystujemy ją w sytuacji, gdzie ilość powtórzeń pętli zależeć będzie od jakiegoś logicznego warunku. Pętla ta pomimo niespełnionego warunku już na samym początku

wykona się zawsze przynajmniej raz, gdyż sprawdzanie warunku odbywa się po wykonaniu kodu pętli.

```
// Różnica pomiędzy while, a do - while polega na tym, że w do
while warunek sprawdzany jest pod koniec obiegu pętli
// co oznacza, że nawet jeżeli warunek jest niespełniony to pętla wykona się przynajmniej raz.
int iterator = 0;
bool warunek = false;
do
{
   iterator++;
} while (warunek);
```

6. Tablice

Tablica jest to specjalny rodzaj zmiennej, w której możemy przechowywać więcej niż jedno wartość.

Deklaracja tablicy:

***typ zmiennych*[] *nazwa tablicy* = new *typ zmiennej* [*wielkość tablicy*];**

```
// tworzenie prostej tablicy na zmienne całkowite:

// zmienna typu tablicowego
int[] tablicaNaLiczbyCalkowite;
// następnie tworzymy tablice z przypisujemy do zmiennej
// w nawiasach musimy podać ile elementów ma przechowywać tablica
tablicaNaLiczbyCalkowite = new int[3];
// do elementów tablicy odwołujemy się po ich indeksie
// Ważne!!! indeksowanie odbywa się od zera,
// to znaczy, że pierwszy element tablicy jest pod indeksem 0.

// przypisywanie wartości elementom tablicy:

tablicaNaLiczbyCalkowite[0] = 1;
tablicaNaLiczbyCalkowite[1] = 2;
```

```
tablicaNaLiczbyCalkowite[2] = 3;
tablicaNaLiczbyCalkowite[3] = 4; // odwołanie się do elementu spoza tablicy spowoduje błąd

//w analogiczny sposób możemy dostać się do wartości zapisanych w tablicy:
int liczbaPodIndeksemjeden = tablicaNaLiczbyCalkowite[1];

// najlepszym sposobem na przeglądanie tablicy są pętle - w tym wypadku najbardziej użyteczna będzie p
ętla for
// każda tablica ma właściwość Length, która zwraca jej długość
int dlugoscTablicy = tablicaNaLiczbyCalkowite.Length;

// należy pamiętać, że indeks ostatniego elementu to : Length - 1;
for (int i = 0; i < dlugoscTablicy; i++)
{
   // wypisuje kolejne elementy z tablicy do zmiennej element
   int element = tablicaNaLiczbyCalkowite[i];
}
```

7. Metody

Metody są to kawałki kodu wykonujące jakąś operację. Do metody można przekazywać parametry różnego typu oraz metoda może, ale nie musi zwrócić wyniku swojego działania.

Definicja metody:

***modyfikator typ nazwaMetody* (*lista argumentów*)**

{

  // ciało metody

}

Modyfikator określa zakres, gdzie metoda będzie dostępna. Możliwe modyfikatory to:

private - dostępna tylko dla klasy, w której została stworzona

protected - dostępna tylko dla klasy, w której została stworzona oraz w klasach pochodnych

internal - dostępna w projekcie, w którym została napisana

public - dostępna w projekcie, w którym została napisana oraz innych bibliotekach

Typ określa jakiego rodzaju dane metoda będzie zwracać np: **int**, **bool**, **double**. Jeżeli metoda nie zwraca nic to typ jest określony jako **void.** Metoda może także przyjmować argumenty, czyli schemat danych jakie będziemy do niej przekazywać. Argumenty podajemy w nawiasie oddzielając je przecinkami.

```
// metoda nic nie zwraca i nie posiada, żadnych parametrów
private void MetodaPierwsza()
{
   // ciało metody
}

// metoda zwraca liczbę całkowitą i nie posiada, żadnych parametrów
public int MetodaDruga()
{
   return 1;
}

// metoda nic nie zwraca i posiada jeden parametr
protected void MetodaTrzecia(int a)
{
   // ciało metody
}

// metoda przyjmuje dwie liczby zmiennoprzecinkowe i zwraca wynik ich dodawania
internal double MetodaCzwarta(double a, double b)
{
   return a + b;
}
```

8. Klasy

Klasa jest to pewnego rodzaju przepis, na podstawie którego tworzy się obiekty. Definiuje on właściwości, czyli zmienne znajdujące się w klasie oraz zachowania tworzonych obiektów za pomocą metod.

Budowa klasy:

modyfikator* class *NazwaKlasy

{

  // ciało klasy

}

Tworzenie obiektu klasy:

***NazwaKlasy* nazwaObiektu = new *NazwaKlasy*();**

Konstruktor - jest to specjalna metoda, która wykonuje się w chwili tworzenia obiektu. Jej zadaniem jest inicjalizacja pól obiektu. Musi nazywać się identycznie jak klasa. Nic nie może zwracać, powinien być publiczny**.**

```
// definicja klasy składa się z modyfikatora dostępu,
// słowa kluczowego class oraz nazwy.
// klasy nazywamy z dużej litery
public class Osoba
{
   // zmienne w klasie nazywamy polami
   // - opisują one właściwości klasy, czyli definiują jej stan
   // przed zmiennymi mogę znajdować się modyfikatory dostępu.
   // Jeśli nie zdefiniujemy modyfikatora to każde pole jest domyślnie prywatne

   // publiczne pole - mamy do niego dostęp z poza klasy
   public string imie;
   public string nazwisko;
```

```
    // prywatne pole - dostęp mają tylko elementy wewnątrz klasy
    // (konstruktory, metody, klasy zagnieżdżone)
    private decimal zarobki;

    // konstruktor - należy go rozumieć jako specjalną metodę w klasie,
    // która nie ma typu zwracanego
    // oraz nazywa się tak samo jak klasa.
    // Jest on wywoływany podczas tworzenia obiektu klasy
    // zazwyczaj wykorzystujemy go do ustawienia jakiś startowych wartości dla kasy.
    public Osoba() // konstruktor bezparametrowy
    {

    }

    public Osoba(string i, string n, decimal z) // konstruktor parametrowy.
        // Przyjmuje on wartości dla naszych pól w klasie i ustawia ich wartości.
    {
        imie = i;
        nazwisko = n;
        zarobki = z;
    }

    // Klasy mogą posiadać także metody, które opisują ich zachowanie/
    // metody także mogą mieć modyfikatory dostępu.
    // Jeżeli nie zdefiniujemy modyfikatora to każda metoda jest domyślnie prywatna.
    public void WpiszDane() // Wypisuje do konsoli wartości pól
    {
        Console.WriteLine($"Osoba {imie} {nazwisko} zarabia {zarobki}");
    }

}
```

```
// Tworzymy zmienną na obiekt naszej klasy
Osoba mojaOsoba;

// tworzymy nowy obiekt typu Osoba i przypisujemy go do zmiennej
mojaOsoba = new Osoba(); // obiekty tworzymy za pomocą słowa kluczowego 'new'
mojaOsoba.imie = "Jan";
mojaOsoba.nazwisko = "Kowalski"; // ustawiamy własności publicznym polą klasy
// nie mamy dostępu do zarobków ponieważ są prywatne


// tworzymy kolejny obiekt typu Osoba, tym razem korzystając z konstruktora z parametrami
Osoba mojaOsoba2 = new Osoba("Michał", "Nowak", 2500m);

mojaOsoba.WpiszDane(); // wywołujemy metodę zdefiniowaną w klasie Osoba
```

9. Przydatne metody

1. Praca z konsolą

```
// wypisanie tekstu na konsole
Console.WriteLine("Witaj świecie")

// pobieranie tekstu z konsoli i zapisanie do zmiennej
string tekst = Console.ReadLine();
```

2. Zamiana wartości tekstowej na typ liczbowy

```
// pobranie wartości z konsoli
string wiekTekstowo = Console.ReadLine();

// zmiana typu tekstowego na typ liczbowy
int wiek = int.Parse(wiekTekstowo);
```

3. Pobieranie tekstu z pliku

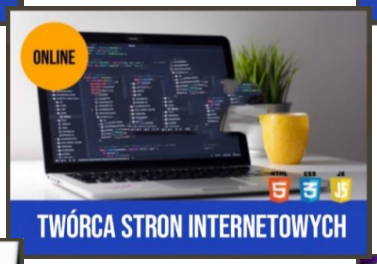
```
// otwiera strumień do odczytu pliku z podanej ścieżki
using (StreamReader sr = new StreamReader("ścieżka_do_pliku"))
{
   // zmienna na pobraną pojedynczą linię tekstu
   string linia;
   // czytamy plik linijka po linijce
   while ((linia = sr.ReadLine()) != null)
   {
      // obsługa pobranych linii tekstu
   }
}
```

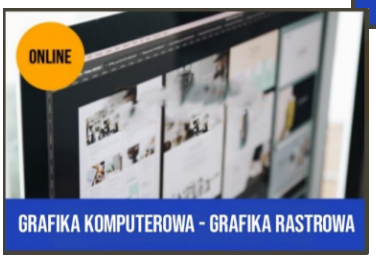
4. Zapis do pliku

```
// otwiera strumień do zapisu do pliku z podanej ścieżki
using (StreamWriter sw = new StreamWriter("ścieżka_do_pliku"))
{
   // zmienna z tekstem do zapisania
   string linia = "Tekst do zapisania";
   // zapisanie wartości zmiennej do pliku
   sw.WriteLine(linia);
}
```

giganci programowania

Chcesz stworzyć jeszcze więcej gier i wypróbować nowe możliwości? Zapraszam Cię na krótkie kursy online, w których możesz uczestniczyć w dowolnym czasie. Nowe gry, kolejne wyzwania - dla każdego, kto w grach i programowaniu chce być mistrzem. Zajrzyj, sprawdź, skorzystaj!





Zapisz się już dziś!

https://www.giganciprogramowania.edu.pl/

giganci programowania

sekretariat@giganciprogramowania.edu.pl

tel: 22 112 10 63