# [KURS C#] 2 + 2



**UWAGA!**

**Nowa wersja kursu dostępna** jest na moim portalu z kursami. Nadal jest on **BEZPŁATNY**! A jednocześnie posiada więcej lekcji, wygodną nawigację i możliwość zapamiętania, w którym miejscu skończyłeś ostatnio!. Znajdziesz go [**POD TYM LINKIEM**](https://kurs-szarpania.pl/nauka/product/szkola-szarpania-intro)!

Aplikacja, którą tworzymy w ramach tego kursu ma być aplikacją bankową dla naszego kolegi Jacka. Skoro aplikacja bankowa to i pieniądze. A jeżeli pieniądze to i liczenie. Tak więc dzisiaj powiemy sobie co nieco o **obliczeniach w języku C#** po to, żeby dać Jackowi możliwość mnożenia pieniędzy.

Ta część będzie bardziej teoretyczna. Tzn. kod jak najbardziej się pojawia jednak nie demonstruję go w odniesieniu do tego co teraz robimy w projekcie. Zachęcam jednak to samodzielnego próbowania na osobnej aplikacji. My zaś tę wiedzę wykorzystamy w późniejszym czasie kiedy całość zacznie nabierać poważnych kształtów.

Nie będziemy też tutaj omawiać wszystkich dostępnych operatorów. Jedynie te, których się używa w miarę często i na pewno Ci się przydadzą.

# Proste jak 2+2

W części poświęconej zmiennym dowiedzieliśmy się, że istnieją w **C#** m.in. **typy liczbowe**. Bo przecież spora część programów opiera się o wszelkiego rodzaju liczby. A skoro są liczby to i jakieś operacje są na nich na pewno możliwe. I tak jest faktycznie.

W języku C#, tak jak w większości języków programowania, mamy do dyspozycji wbudowane podstawowe **operacje arytmetyczne**. Ponieważ wszystkie obliczenia prędzej czy później sprowadzają się do takich prostych operacji. I to właśnie takie działania potrafi wykonać **procesor** naszego komputera.

## Operatory arytmetyczne

To czym się tutaj teraz zajmiemy to są **operatory arytmetyczne**. Chodzi np. o dodawanie, mnożenie itd. Nie przedłużając – poniżej masz listę najpotrzebniejszych operatorów, które powinieneś pamiętać.

W przykładach używane litery mogą być zarówno zmiennymi jak i wpisanymi na sztywno liczbami, albo funkcjami, które zwracają jakąś liczbę. Pokazane operacje możemy sobie wyobrazić jako funkcje, które zwracają wartość w postaci wyniku obliczeń, więc taką wartość możemy potem gdzieś **przypisać** czy użyć jako **argument funkcji**:

* a + b  najpopularniejszy operator czyli po prostu **dodawanie**. a dodajemy do b .
* Analogicznie a – b jest podobne jak powyżej z tym, że teraz **odejmujemy** b od a .
* a \* b  to znane wszystkim **mnożenie**.
* a / b  **dzielenia** też raczej nikomu nie trzeba przedstawiać. Jednak tutaj jest jeden dosyć istotny haczyk. Jeśli przypomnimy sobie część o typach zmiennych to powiedziałem tam, że są typy dla **liczb całkowitych** i **z przecinkiem**. I teraz jeżeli dzielimy liczby z przecinkiem to sprawa jest raczej jasna i otrzymamy to czego oczekujemy. Jednak jeżeli podzielimy liczby całkowite to **wynik** też będzie **całkowity**, część dziesiętna zostanie obcięta. Przykładowo 6 / 4 da wynik 1 , a nie 1.5 jeżeli będziemy używali **liczb całkowitych**. Chcąc wykonać dzielenie i dostać wynik z wartością po przecinku **przynajmniej jedna z liczb musi być liczbą z przecinkiem**. W naszym przypadku wynik 1.5 otrzymamy pisząc 6.0 / 4  albo 6 / 4.0  albo 6.0 / 4.0 . Jeżeli nie korzystamy ze stałych liczb tylko ze zmiennych to zasada jest taka sama – chcąc dostać wynik z przecinkiem przynajmniej jedna użyta zmienna musi być zmienną **typu zmiennoprzecinkowego**.
* a % b tutaj mamy co takiego jak **reszta z dzielenia**. Łączy się to z **dzieleniem liczb całkowitych**. W przedstawionym powyżej przykładzie z dzieleniem 6 / 4 resztą takiego dzielenia będzie 2 . Jest to różnica pomiędzy dzielną, a wynikiem dzielenia pomnożonym przez dzielnik, czyli po prostu to co zostało jako wartość mniejsza niż dzielnik.

## Kolejność działań



Operatory arytmetyczne zachowują się podobnie jak te używane na matematyce, tzn. zachowują **kolejność wykonywania działań**. Bo jeszcze nie wspomniałem, ale operacje na liczbach możemy składać w **dłuższe wyrażenia** gdzie pojawia się **więcej operatorów**.

Przykładowo możemy zapisać działanie jako a + b \* c – d bez konieczności rozbijania go na osobne linijki, w których wykonywalibyśmy **pojedyncze działania**. I w takim przypadku kolejność wykonywania działać zostanie zachowana, tzn. najpierw wykona się **mnożenie**, a dopiero potem **dodawanie** i **odejmowanie**.

**Działania równoważne**, czyli chociażby dodawanie i odejmowanie, albo mnożenie i dzielenie wykonują się od **lewej do prawej**.Ale tak jak w matematyce możemy też tutaj zmienić ich kolejność dodając **nawiasy**.

Tak więc chcąc aby najpierw wykonały się działania dodawania i odejmowania, a dopiero ich wyniki zostały pomnożone skorzystamy z zapisu (a + b) \* (c – d) .

## 2 w 1

Poza tymi oczywistymi operatorami, które znamy też z matematyki, mamy w języku C# dostępne też operatory, które łączą w sobie **operację matematyczną i przypisanie**. Przydają się one w takich przypadkach jak zwiększenie jakieś wartości o podają liczbę.

Poniżej ich lista:

* Wyrażenie a += b oznacza to samo co a = a + b
* Zapis a -= b jest tym samym co a = a – b
* Taki zapis a \*= b to nic innego jak a = a \* b
* a /= b znaczy tyle co a = a / b
* Ten operator a++ oznacza a = a + 1 i nazywa się **postimkrementacją**
* A a– jest tym samym co a = a – 1 i nazywa się **postdekrementacją**

Fakt, że wykonujemy tutaj też **przypisanie** powoduje, że nasze a musi być w tym wypadku **zmienną**, do której możemy coś przypisać. Tak więc nie ma możliwości zapisu 5 += 3 bo nie możemy zmienić wartości liczby 5 .

Dodatkowo w przypadku operacji zwiększania lub zmniejszania o 1 trzeba zapamiętać **kolejność działań** jeżeli używamy tego operatora np. dla argumentu funkcji.

W przypadku a++ i a– najpierw zostanie **przekazana wartość**, a dopiero potem wartość zmiennej się zmieni. Dlatego w tym kodzie:

int a = 2;

Calc(a++);

Console.WriteLine(a);

Funkcja calc()  otrzyma wartość **2**, a potem na ekranie zostanie wypisana wartość **3**. Dzieje się tak, ponieważ najpierw zwracana jest wartość zmiennej, a dopiero po tym wykonywane jest **zwiększenie o 1**.

Jeżeli byśmy chcieli odwrócić tą kolejność to możemy skorzystać z operatora **preinkrementacji**/**predekrementacji**. Wtedy zapis zmienia się z a++ na ++a . I teraz zmieniając nasz kod:

int a = 2;

Calc(++a);

Console.WriteLine(a);

Zarówno funkcja calc()  dostanie wartość **3** jak i wartość **3** zostanie wyświetlona na ekranie. Bo zwiększenie o 1 wykona się wcześniej niż przekazanie wartości do funkcji calc() .

# Liczymy

Większość **działań na liczbach** w języku **C#** sprowadza się do przedstawionych powyżej. Na pewno z nich skorzystamy w naszej aplikacji bankowej. Dzięki temu klienci Jacka będą mogli wykonywać wszelkie operacje finansowe na swoich kontach.

[**Poprzednia lekcja – zrobisz coś dla mnie?**](https://zajacmarek.com/kurs-c-zrobisz-cos-dla-mnie/)

[**Następna lekcja – genetyka klas**](https://zajacmarek.com/kurs-c-genetyka-klas/)

[**Spis treści**](https://zajacmarek.com/kurs-programowania-c-sharp/#chapters)

## 5 komentarzy

* Kamil pisze:

[20 listopada 2019 o 23:42](https://zajacmarek.com/kurs-c-2-2/#comment-21644)

Nie rozumiałem wyjaśnienia w operatorach a % b i chwilę mi zajęło aby do tego dojść, ale powinno być np. tak:

**a % b**

Jest to wartość po przecinku wyniku działania a / b pomnożona przez dzielnik. Lub:

Różnica ilorazu działania a / b oraz liczby całkowitej tego ilorazu, pomnożona przez wartość dzielnika.

Ponadto wynik jest zaokrąglony do liczby całkowitej.

Chwilę mi zajęło aby do tego dojść, ponieważ nie rozumiałem wyjaśnienia. Jeżeli źle do tego doszedłem, proszę mnie poprawić.

Dodatkowo bym prosił o przykład wykorzystania tego %. Dopiero się uczę i nic mi do głowy nie przychodzi.