**[KURS C#] Pod warunkiem, że…**



**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)!

W życiu rzadko jest tak, że mamy **jedną prostą ścieżkę**, której się zawsze trzymamy. Również w banku pewne operacje mogą dać różne efekty w zależności od kilku czynników.

Przykładowo próba wypłaty pieniędzy skończy się sukcesem jeżeli mamy na koncie odpowiednio dużo środków. Ale jeżeli tych środków brakuje to dostaniemy odpowiedni komunikat w bankomacie, albo odmowę w okienku w placówce banku.

Całe życie to **podejmowanie decyzji**. Dlatego również w programowaniu podejmowanie decyzji jest obecne i niekiedy niezbędne.

**Prawa? Fałsz?**

O ile w prawdziwym życiu różne decyzje podejmujemy w sposób nie zawsze jednoznaczny i konkretny, wahamy się, kierujemy emocjami itd. o tyle w programowaniu jest prościej. Albo jakiś **warunek** jest spełniony, albo nie.

Dlatego nieważne nad czym nasz program będzie się zastanawiał – i tak ostatecznie sprowadzi się to do szeregu odpowiedzi *„tak”*, *„nie”*, *„tak”*, *„nie”*, *„nie”*, *„tak”*. Bo komputer działa na skrajnościach. Coś jest albo **prawdą** albo **fałszem**. Na tej podstawie podejmuje decyzje.

Przykład z wypłacaniem pieniędzy jest tutaj dobry. Kiedy chcesz je wypłacić to powstaje pytanie *„czy mam tyle pieniędzy na koncie?”*. I jedyne odpowiedzi to **tak** lub **nie**. Jeżeli nie mam to mogę dostawać kolejne pytania, np. czy dostanę pożyczkę – **tak** lub **nie**? itd. I w taki sposób będziemy też wybierać ścieżki, którymi podążać będzie nasz kod.

**Pytanie o prawdę**

W porządku, skoro już wiemy o co chodzi z decyzjami w programowaniu to czas przejść do najważniejszego – jak mamy w programie **podjąć decyzję**?

Sprawa jest względnie prosta. Przynajmniej jeżeli chodzi o strukturę podejmowania decyzji. Ogólnie wygląda to tak, że w naszej funkcji wstawiamy fragment, który brzmi mniej więcej tak: *„Jeżeli x jest prawdzie to zrób…”* i dajemy blok kodu, który zadziała tylko jeżeli ten **x** jest faktycznie **prawdą**. W języku **C#** służy do tego **instrukcja** if:

if (x)

{

// Jeżeli x jest prawdą

}

Ale czasami jeżeli jeden **warunek** nie jest spełniony to może inny jest. Dlatego możemy stwierdzić, że *„jeżeli x nie jest prawdziwe to jeżeli y jest prawdziwe to zrób…”*.

Do tego służy kolejna **instrukcja** else if, która jest bezpośrednio związana z tą pierwszą:

if (x)

{

// Jeżeli x jest prawdą

}

else if (y)

{

// Jeżeli x NIE jest prawdą, ale y jest prawdą

}

else if (z)

{

// Jeżeli x NIE jest prawdą i y NIE jest prawdą, ale z jest prawdą

}

Kolejnych warunków może być więcej niż jeden. Ważne, że muszą być połączone z samą instrukcją if , która jest tym pierwszym.

Ale jest też tak, że chcemy, żeby coś się zadziało kiedy żaden z warunków nie jest prawdą. Np. ktoś nam mówi *„chcę założyć konto rodzinne”*. Więc sprawdzamy *„czy konto rodzinne to konto oszczędnościowe? Nie. Czy konto rodzinne to konto rozliczeniowe? Nie. Ok, nie mamy takiego konta”* i odpowiadamy klientowi *„Niestety nie mamy takiego konta w ofercie”*. Do wykonania czegoś co ma się zadziać jeżeli **wszystkie warunki** nie zostały spełnione służy instrukcja else, którą zawsze umieszczamy na końcu, i która może być **tylko jedna** dla jednej instrukcji if:

if (x)

{

// Jeżeli x jest prawdą

}

else

{

// Jeżeli x NIE jest prawdą

}

Jeśli mamy też alternatywne opcje to else również ląduje na końcu:

if (x)

{

// Jeżeli x jest prawdą

}

else if(y)

{

// Jeżeli x NIE jest prawdą, ale y jest prawdą

}

else

{

// Jeżeli x NIE jest prawdą i y NIE jest prawdą

}

**Gdzie ukryta jest prawda?**

Skoro wiemy już jak sprawdzić czy coś jest prawdą czy nie to powstaje pytanie co możemy w ten sposób sprawdzać?

Najbardziej ogólna odpowiedź jest taka, że w **instrukcji warunkowej** (bo takiego typu instrukcje tutaj mamy) możemy sprawdzać cokolwiek co w rezultacie daje wartość **prawda/fałsz**. Chodzi więc o wszystko co na końcu daje wynik typu **bool**, bo właśnie **bool** jest **typem danych**, który przechowuje wartość **true/false** (**prawda/fałsz**).

Tak więc pierwsze co na pewno przychodzi do głowy to funkcja, która zwraca taką wartość:

bool isTrue()

{

return true;

}

bool isFalse()

{

return false;

}

Ale w takiej postaci jest to trochę bez sensu bo jednak nie chcemy na sztywno wrzucać wartość w kod.

Wtedy niepotrzebna by nam była instrukcja **if**, bo przecież znając wynik w trakcie pisania kodu moglibyśmy po prostu zostawić tylko te fragmenty kodu, które się wykonają przy takiej wartości.

Musi więc istnieć coś co pozwoli nam „produkować” **wartości logiczne**. I faktycznie tak jest. Są to **porównania**.

**Porównajmy się**

Najbardziej oczywistym porównaniem jest po prostu sprawdzenie czy dwie **wartości są takie same**. Możemy taką operację wykonać dla liczb, co oczywiste, ale też np. dla tekstów. A właściwie to dla każdego obiektu, jednak w tym momencie jest to temat wchodzący zbyt głęboko. Dlatego zostańmy przy **liczbach**, **tekstach** i po prostu zmiennych typu **bool**.

**Porównanie** wykonuje się za pomocą **podwójnego znaku równości**.

Tak więc jeżeli mamy dwie wartości to po prostu porównujemy je w ten sposób:

a == b

Możemy porównywać zarówno dwie zmienne, dwie wpisane na stałe wartości albo zmienną i stałą wartość:

int a = 1;

int b = 2;

a == b

a == 1

2 == 2

Przy porównywaniu tekstu ważne żeby pamiętać, że porównywana jest też wielkość liter, więc teksty muszą być faktycznie identyczne.

Skoro można sprawdzić czy dwie wartości są równe to przydałoby się też sprawdzić czy są różne, prawda? I również to jest możliwe. Służy do tego **operator nierówności**. Różni się tym, że zamiast pierwszego znaku równości ma wykrzyknik:

a != b

Powyższy zapis oznacza *„czy a nierówna się b?”*.

Oba te porównania zwracają w rezultacie wartość true lub false, czyli po prostu wartość typu **bool**. Także nic nie stoi na przeszkodzie, żeby wynik tego porównania wstawić do zmiennej:

bool isEqual = a == b;

Od teraz w zmiennej isEqual znajdzie się wartość **true** lub **false**, w zależności do tego czy wartości **a** i **b** będą **równe** czy nie.

Potem taką wartość możemy oczywiście wykorzystać w **kolejnym porównaniu** jako parametr **a**lub **b** z powyższego przykładu.

**Ja mam większe**

Poza sprawdzeniem czy wartości są równe albo różne możemy również wykonać porównanie typowe dla liczb. Mianowicie można sprawdzić czy jedna wartość jest większa bądź mniejsza od drugiej.

* Czy wartość **a** jest większa od **b**: a > b
* A może wartość **a** jest mniejsza od **b**: a < b
* Czy wartość **a** jest większa bądź równa **b**: a >= b
* Albo czy wartość **a** jest mniejsza bądź równa **b**: a <= b

**Sprawdźmy to!**

Jak w takim razie wygląda wykorzystanie porównania w kontekście **instrukcji if**? Bardzo prosto. Skoro wiemy już jakie porównania możemy zastosować to nie zostaje nam nic innej jak wstawienie ich w **instrukcji if** właśnie:

int a = 2;

int b = 4;

if (a >= b)

{

Console.WriteLine("a jest większe lub równe b");

}

else

{

Console.WriteLine("b jest większe od a");

}

W tym przypadku w konsoli dostaniemy tekst *„b jest większe od a”*. Dlatego, że warunek a >= b **nie jest prawdą**, a więc nie wchodzimy w blok w instrukcji **if**.

W przypadku takim jak powyżej porównanie może nie mieć sensu. Jednak wyobraźmy sobie, że jedna z wartości będzie zależała od tego co wpisze użytkownik. Wtedy już nie możemy z góry przewidzieć czy warunek będzie prawdą czy nie. Wartości mogą też zależeć od jakichś obliczeń i innych tego typu operacji. Wtedy takie porównanie ma sens i pozwala decydować, który fragment kodu chcemy wykonać.

**Więcej niż jeden warunek to…?**

Pokazane przed chwilą przykłady są już bardzo użyteczne. Jednak życie nie zawsze jest tak proste. Nieraz zdarza się, że jakaś decyzja musi być podjęta na podstawie **więcej niż jednego porównania**. I co wtedy? Czy jesteśmy skazani na zapis podobny do poniższego?

if (a == b)

{

if (b == c)

{

Console.WriteLine("a, b i c są równe");

}

}

Absolutnie nie! W języku **C#** jak najbardziej można **warunki logiczne** łączyć ze sobą. Możemy do tego użyć operatorów && lub ||.

Pierwszy z nich oznacza **„i”**, czyli cały warunek będzie prawdziwy jeżeli warunki po obu stronach tego łącznika będą prawdziwe. Drugi z nich to operator **„lub”**, który zwraca prawdę wtedy kiedy przynajmniej jeden z warunków jest prawdziwy.

Ale może to brzmieć zawile więc czas na przykład. Komentarze w kodzie będą mówiły jakie „zdanie” zostało sprawdzone w **operatorze if**:

if (a == b && b == c)

{

// Jeżeli a jest równe b i b jest równe c

}

if (a == b || b == c)

{

// Jeżeli a jest równe b lub b jest równe c

}

Nic nie stoi na przeszkodzie, żeby tych połączeń było więcej. Należy jednak wtedy pamiętać, że operator **&&** ma wyższy priorytet niż operator **||**. A więc w poniższym zapisie:

if (a == b && b == c || c == d)

{

}

Sprawdzamy czy **a** jest równe **b** i **b** jest równe **c** lub samo **c** jest równe **d**. To znaczy, że wystarczy aby **c** było równe **d** i cały warunek będzie prawdziwy ponieważ taki zapis jest równy temu:

if ((a == b && b == c) || c == d)

{

}

Czyli najpierw będzie sprawdzone to co jest w nawiasie, a dopiero potem wynik tej części zostanie zastosowany w operatorze **„lub”**.

**Nie zgadzam się!**

Ostatnia rzecz, o której tutaj wspomnimy to **zaprzeczenie**.

Zdarzają się sytuacje kiedy chcemy aby warunek był spełniony (czyli dał ostatecznie wartość true) kiedy wszystkie porównania czy rezultaty jakichś funkcji dają false. Możemy w tym celu zastosować **operator negacji**, który w kodzie jest reprezentowany przez **wykrzyknik**. Jego działanie jest proste – jeżeli postawimy go **przed operacją** dającą **prawdę** to zwróci **fałsz**, a jeśli operacja będzie dawała **fałsz** to on ostatecznie zwróci **prawdę**.

Przykładowo:

int a = 2;

int b = 3;

if (!(a == b))

{

// a nie jest równe b

}

bool x = false;

if (!x)

{

// !x jest równe true

}

I to tyle. Powyżej jest przykład zarówno dla operatora porównania jak i dla zmiennej przechowującej wartość logiczną.

Operator negacji jest bardzo prosty, jednak znajduje swoje zastosowanie. Zwłaszcza kiedy wartość logiczną otrzymujemy z funkcji, a chcielibyśmy aby warunek zadziałał nie w przypadku gdy zwróci ona prawdę, ale w przeciwnym. Wtedy też zamiast pisać:

if (func() == false)

{

}

Możemy użyć operatora negacji i zapisać to w postaci:

if (!func())

{

}

**Trudny wybór**

Możliwość wybierania **różnych wariantów wykonania kodu**, w zależności od aktualnego stanu aplikacji albo tego co podał użytkownik to niewątpliwie duża rzecz. Również w naszym systemie bankowym znajdzie ona swoje miejsce. Dzięki temu będziemy mogli na bieżąco reagować na to co robi użytkownik i na jakich danych aktualnie pracujemy. Praktyczne zastosowanie instrukcji warunkowych już za moment.

A teraz czas zacząć gromadzić więcej danych!