Materiały powtórzeniowe

Lekcja 9-10.

Pętle

Witaj w materiałach powtórzeniowych przygotowanych specjalnie dla Ciebie. Znajdziesz tutaj powtórkę materiału z ostatniej lekcji oraz zadania do wykonania dla chętnych. Z pewnością sobie poradzisz. Powodzenia!

Cel lekcji

Celem lekcji jest zapoznanie się z pojęciem pętli.

W programie komputerowym często zachodzi potrzeba wykonania wielu bardzo podobnych do siebie działań. Przykładem takich operacji może być wyświetlenie kilku kolejnych liczb, obliczenie wartości dowolnej liczby lub oczekiwanie na naciśnięcie przez użytkownika konkretnego znaku na klawiaturze. Aby wypisać kolejne liczby naturalne (np. z przedziału od 1 do 9) można skorzystać z 9 poleceń *Console.WriteLine().* Należy zauważyć jednak, że te polecenia będą się od siebie różnić tylko jednym elementem – wartością wyświetlanej liczby. Gdyby zaś rozszerzyć zakres wyświetlanych wartości np. do 1000, nie trzeba nikogo przekonywać, że pisanie w tym celu 1000 niemal identycznych linii jest po prostu stratą czasu i energii (zarówno programisty jak i komputera). Dlatego też w każdym języku programowania występują pętle – konstrukcje, które korzystając ze stosunkowo prostego zapisu kodu umożliwiają wielokrotne wykonanie określonego zbioru poleceń.

Pętla FOR

W pętli *for* występuje zmienna, która nazywana jest licznikiem pętli. Dzięki tej zmiennej istnieje możliwość określania, ile razy ma się wykonać pętla (czyli liczbę przebiegów pętli). Oprócz licznika pętli, duże znaczenie ma wyrażenie logiczne, które pozwala zadecydować, czy pętla się wykona czy też nie oraz polecenia wykonywane w samej pętli.

Składnia pętli FOR:

```
for (od kiedy; do kiedy; co ile)
{
    // ciało pętli (powtarzane instrukcje)
}
```

gdzie:

- od kiedy, oznacza stan początkowy pętli, od czego zaczynamy
- do kiedy, określa warunek działania pętli, gdy przestaje być spełniony to pętla się zatrzymuje
- co ile - co ma się dziać co iterację pętli

w skrócie: *for(od linii startu; aż miniesz linię mety; zrób kolejny krok).*

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

Przykład z życia wzięty - Załóżmy, że biegacz startuje w biegu na 100 metrów. Linia startu to punkt o wartości 0. Koniec biegu, to moment minięcia mety czyli, gdy przebiegliśmy 100 metrów (do tego momentu musimy przebierać nogami). Pomiędzy początkiem a końcem, ilość przebiegniętych metrów rośnie – załóżmy, że aktualizujemy ją co jeden metr.

Algorytm w tym wypadku wygląda tak:

1. Stoję na zerowym metrze.

2. Sprawdzam, czy jestem już na mecie

3. Jeżeli nie, to przesuwam się do przodu o metr i wracam do punktu 2

4. Jeżeli tak, kończę pętlę

Przerabiając to na kod, dostaniemy coś takiego:

```
for (int i = 0; i <= 100; i++)
{
    Console.WriteLine($"to już {i} metr")
}
```

Pętla WHILE

Przejdźmy do omówienia kolejnych pętli. Zapis pętli ***while*** jest prostszy od pętli ***for***.

Składnia:

```
while (wyrażenie logiczne)
{
  //ciało pętli (instrukcje)
}
```

Algorytm pętli *while* można przedstawić w następujący sposób:

- jeżeli wyrażenie logiczne ma wartość *true*, to wykonaj instrukcje zawarte w pętli. Powtarzaj ten krok, dopóki jest spełnione wyrażenie logiczne,
- jeżeli wyrażenie logiczne zwraca wartość *false* – zakończ działanie pętli.

Przykład:

```
int i = 0;
while (i < 10)
{
    Console.WriteLine("Cyfra " + i);
    i++;
}
```

UWAGA! Choć pętle w programowaniu są potężnym narzędziem, które pozwala nam wykonywać te same operacje wielokrotnie, to musimy być ostrożni, aby nie stworzyć nieskończonej pętli, która nigdy się nie zakończy.

Nieskończona pętla to taka, która nigdy nie przestaje działać, ponieważ warunek zakończenia pętli nigdy nie zostaje spełniony. Może to spowodować, że program "zawiesi się", a komputer przestanie reagować na inne polecenia.

Pamiętaj: Tworzenie nieskończonej pętli może spowodować, że Twój program zapętli się, co może zmusić Cię do jego ręcznego zatrzymania. Zawsze dokładnie przemyśl, jak ma działać Twoja pętla i upewnij się, że istnieje warunek jej zakończenia!

Jak unikać nieskończonych pętli?

1. Zawsze sprawdzaj warunek pętli: Upewnij się, że warunek, który określa, kiedy pętla powinna się zakończyć, jest poprawnie zdefiniowany.
2. Pamiętaj o aktualizacji licznika: Jeśli używasz pętli, która zależy od zmiennej (np. for lub while), upewnij się, że zmienna ta zmienia się w każdej iteracji, aby mogła spełnić warunek zakończenia.
3. Testuj swoje pętle: Zawsze sprawdzaj, czy pętla kończy się, jak zamierzałeś, aby uniknąć nieskończonych iteracji.

Pętla DO….WHILE

Sposób działania pętli ***do..while*** jest bardzo podobny do działania pętli *while*. Tu także występuje wyrażenie logiczne, od którego zależy, czy pętla będzie wykonana (dla wartości *true*), czy też zostanie zakończona (dla wartości *false*). Różnica pomiędzy tymi pętlami polega na tym, że w pętli ***while*** warunek logiczny sprawdzany jest **przed pierwszym** wykonaniem pętli, a w ***do..while*** warunek logiczny testowany jest **po pierwszym** wykonaniu pętli. **W konsekwencji pętla *do..while* wykonana zostanie przynajmniej jeden raz, w czasie gdy pętla *while* może nie zostać wykonana w ogóle, gdyby od wyrażenie logiczne było nie spełnione.**

Składnia pętli *do – while:*

```
do
{
// ciało pętli (instrukcje)
}
while (wyrażenie logiczne)
```

Przykład:

```
static void Main(string[] args)
{
    string odpowiedz;
    do
    {
        Console.WriteLine($"Czas: {DateTime.Now}",);
        Console.WriteLine("Ponownie pokazać aktualny czas? (t/n)");
```

```
        odpowiedz = Console.ReadLine();
    } while (odpowiedz != "n");
}
```

Dodatkowa misja Programisty:

Zadanie 1
Używając dowolnej pętli napisz program wypisujący w konsoli 10 pierwszych wielokrotności liczby 3.

Zadanie 2
Używając pętli while napisz program, który metodą naiwną (algorytmem Euklidesa) będzie sprawdzać największy wspólny dzielnik (NWD) dwóch podanych przez użytkownika liczb.

Podpowiedz: https://pl.wikipedia.org/wiki/Algorytm_Euklidesa

Przykładowe wyniki: NWD(84, 126) = 42, NWD(150, 120) = 30, NWD(70, 28) = 14

Zadanie 3
Napisz program, generujący losowe cd-keys (klucz do oprogramowania) o podanej przez użytkownika liczbie znaków. Klucz ma się składać z bloków po 5 znaków rozdzielonych znakiem -.

Podpowiedź: Losowanie możemy zaimplementować w następujący sposób:

```
Random maszynaLosujaca = new Random();
int cyfra = maszynaLosujaca.Next(0, 10);
klucz += cyfra.ToString();
```