Podstawy programowania

Laboratorium

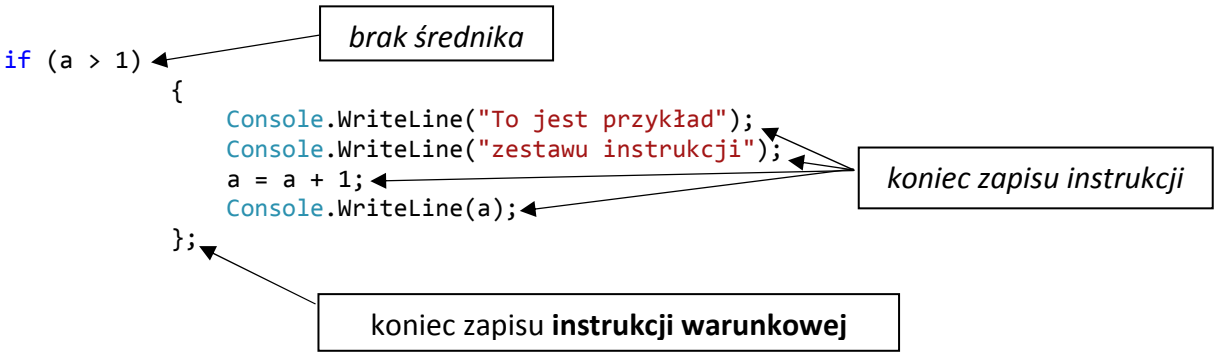
Ćwiczenie 2 – Programowanie strukturalne – podstawowe rodzaje instrukcji

Instrukcja warunkowa if

Format instrukcji warunkowej – **Przykład 1**.

```
if (warunek)
{instrukcja albo zestaw instrukcji};
```

Jeżeli warunek jest spełniony (tzn. warunek jest prawdą) to zostanie wykonana instrukcja lub zestaw instrukcji zapisany w klamrach. Jeżeli warunek nie jest spełniony, instrukcja lub zestaw instrukcji nie zostanie wykonany, a komputer przejdzie do wykonywania kolejnej instrukcji na liście programu po instrukcji warunkowej.
Uwaga! Za nawiasem kończącym warunek nie należy stawiać średnika – obowiązkowy średnik kończący zapis instrukcji warunkowej znajduje się dopiero za klamrą kończącą zestaw instrukcji. Natomiast wewnątrz zestawu instrukcji stosuje się średniki kończące poszczególne instrukcje zestawu, jak w poniższym przykładzie:



Format instrukcji warunkowej – **Przykład 2**.

```
if (warunek)
{instrukcja_1 albo zestaw_instrukcji_1}
else
{instrukcja_2 albo zestaw_instrukcji_2};
```

Jeżeli warunek jest spełniony (tzn. warunek jest prawdą) to zostanie wykonana instrukcja_1 lub zestaw_instrukcji_1 zapisany w klamrach. W przeciwnym przypadku, tzn. jeżeli warunek nie jest spełniony, zostanie wykonana instrukcja_2 lub zestaw_instrukcji_2 zapisany w klamrach.

Uwaga 1! Za nawiasem kończącym warunek nie należy stawiać średnika.
Uwaga 2! Za klamrą kończącą zestaw_instrukcji_1 nie należy stawiać średnika, ponieważ obowiązkowy średnik kończący zapis instrukcji warunkowej znajduje się dopiero za klamrą kończącą zestaw_instrukcji_2.

Jeżeli przykładowy warunek ma postać `(a<1)`to warunek przeciwny do tego warunku ma postać `(a>=1)`, tzn. „a” jest większe lub równe 1.

Format instrukcji warunkowej – **Przykład 3**.

```
if ((warunek_1) & (warunek_2))
{instrukcja albo zestaw instrukcji};
```

Jeżeli warunek_1 jest spełniony i jednocześnie warunek _2 jest spełniony to zostanie wykonana instrukcja lub zestaw instrukcji.

Uwaga! Zapis głównego warunku funkcji warunkowej jest zapisany w nawiasie (nadrzędnym). Poszczególne warunki składowe (warunek_1 oraz warunek_2) zostały zapisane w dodatkowych nawiasach.

Przykłady operatorów:

< - mniejszy
> - większy
<= - mniejszy lub równy
>= - większy lub równy
== - równy
!= - różny
& - i (koniunkcja)
^ - lub (alternatywa)

Tworzenie algorytmu

Polecenie 1:
Na osobnej kartce papieru zapisz (np. słownie) algorytm obliczania (kolejność wykonywania czynności) przy wyznaczaniu pierwiastków x_0, x_1, x_2, równania kwadratowego $y=ax^2+bx+c$. Wypisz nazwy (symbole) wszystkich zmiennych i parametrów wykorzystywanych w ramach tego algorytmu. Wskaż które zmienne mają charakter danych wejściowych, danych wyjściowych oraz danych pomocniczych (o ile występują) w ramach całego procesu obliczeniowego. Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Polecenie 2:
Napisz program obliczający pierwiastki równania kwadratowego. Użytkownik programu powinien definiować postać funkcji poprzez wprowadzenie z klawiatury wartości współczynników a, b, c. Program powinien uwzględniać różne możliwości rozwiązań (jedno, dwa lub zero miejsc zerowych) i wyświetlać pierwiastki z dokładnością do 3 miejsc po przecinku. Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Instrukcja wyboru switch..case

Format instrukcji wyboru:

```
switch (wyrażenie)
{
      case wartosc_1:
         {
           instrukcja_1 albo zestaw_instrukcji_1
         }
           break;
      case wartosc_2:
         {
          instrukcja_2 albo zestaw_instrukcji_2
         }
           break;
      default:
         {
          instrukcja_3 albo zestaw_instrukcji_3
         }
           break;
};
```

Jeżeli wyrażenie (zmienna) ma wartość_1 to zostanie wykonana instrukcja_1 (zestaw_instrukcji_1) zapisana w klamrach. Jeżeli wyrażenie (zmienna) ma wartość_2 to zostanie wykonana instrukcja_2 (zestaw_instrukcji_2) zapisana w klamrach. Jeżeli wyrażenie (zmienna) ma jakąkolwiek inną wartość to zostanie wykonana instrukcja_3 (zestaw_instrukcji_3) zapisana w klamrach.

Polecenie 3:

Przetestuj działanie programu, którego listing zapisano poniżej:

```
Int32 a;
            Console.WriteLine("Podaj liczbę");
            a = Int32.Parse(Console.ReadLine());
            switch (a)
            {
                case 0:
                    {
                        Console.WriteLine("Zero");
                    }
                    break;
                case 1:
                    {
                        Console.WriteLine("Jeden");
                    }
                    break;
                case 2:
                    {
                        Console.WriteLine("Dwa");
                    }
                    break;
                default:
                    {
                        Console.WriteLine("Nie wiem");
                    }
                    break;
            };
```

Polecenie 4:
Napisz program obliczający pierwiastki równania kwadratowego. W celu identyfikacji liczby pierwiastków równania wykorzystaj instrukcję warunkową if, natomiast operacje i obliczenia związane z konkretną liczbą pierwiastków wykonaj korzystając z instrukcji wyboru switch…case. Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Instrukcje iteracyjne (pętle)

Prawidłowe użycie instrukcji iteracyjnej wymaga zdefiniowania czterech podstawowych elementów:

1. **Wartość początkowa** zmiennej sterującej pętlą,
2. **Warunek kontynuacji** (zakończenia) realizacji pętli,
3. **Inkrementacja** (lub dekrementacja) wartości zmiennej sterującej pętlą,
4. **Zestaw instrukcji** do wykonania w ramach działania pętli.

Pętla FOR

Format pętli for:

```
for (wartosc_poczatkowa; warunek_kontynuacji; inkrementacja)
{
     Instrukcja albo zestaw_instrukcji;
};
```

Instrukcja albo zestaw_instrukcji zostanie wykonany wielokrotnie, począwszy od wartości zmiennej sterującej równej wartosc_poczatkowa do momentu, w którym spełniony jest warunek_kontunuacji. Każdorazowo po wykonaniu instrukcji lub zestawu_instrukcji wartość zmiennej sterującej pętlą wzrasta (lub maleje) zgodnie z zapisem inkrementacji (dekrementacji).

Przykładowy zapis:

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

W podanym przykładzie, instrukcja wyświetlająca wartość zmiennej „a” zostanie wykonana w sumie 11 razy, tzn. dla kolejnych wartości zmiennej „a” od 0 do 10 włącznie.

Polecenie 5:
Napisz program, który za pomocą instrukcji FOR dla danych wartości x zmieniających się w przedziale od 1 do 20 obliczy wartości funkcji y=2x+1. Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Pętla DO…WHILE

Format pętli do…while:

```
wartosc_poczatkowa;
do
{
     Instrukcja albo zestaw_instrukcji;
     Inkrementacja;
}
while (warunek_kontynuacji);
```

Uwaga 1! Zwróć uwagę, że średnik kończący zapis pętli znajduje się dopiero za nawiasem kończącym zapis warunku kontynuacji pętli.
Uwaga 2! Zwróć uwagę, że zapis wartości początkowej zmiennej sterującej znajduje się poza formalnym zapisem pętli.

Przykładowy zapis:

```
a = 0;
do
{
    Console.WriteLine(a);
    a++;
}
while (a <= 10);
```

W podanym przykładzie, instrukcja wyświetlająca wartość zmiennej „a" zostanie wykonana w sumie 11 razy, tzn. dla kolejnych wartości zmiennej „a" od 0 do 10 włącznie.

Polecenie 6:

Napisz program, który za pomocą instrukcji DO…WHILE dla danych wartości x zmieniających się w przedziale od 1 do 20 obliczy wartości funkcji y=2x+1. Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Polecenie 7:

Korzystając z programu opracowanego w ramach Polecenia 6 przenieś zapis inkrementacji z końca na początek zestawu_instrukcji. Czy lokalizacja zapisu inkrementacji ma wpływ na działanie pętli?

Pętla WHILE

Format pętli while:

```
wartosc_poczatkowa;
while (warunek_kontynuacji)
{
     Instrukcja albo zestaw_instrukcji;
     Inkrementacja;
};
```

Uwaga 1! Zwróć uwagę, że średnik kończący zapis pętli znajduje się dopiero za klamrą kończącą zapis zestawu instrukcji.
Uwaga 2! Zwróć uwagę, że zapis warunku początkowego znajduje się poza formalnym zapisem pętli.

Przykładowy zapis:

```
a = 0;
while (a <= 10)
{
    Console.WriteLine(a);
    a++;
};
```

W podanym przykładzie, instrukcja wyświetlająca wartość zmiennej „a" zostanie wykonana w sumie 11 razy, tzn. dla kolejnych wartości zmiennej „a" od 0 do 10 włącznie.

Polecenie 8:
Napisz program, który za pomocą instrukcji WHILE dla danych wartości x zmieniających się w przedziale od 1 do 20 obliczy wartości funkcji y=2x+1. Na czym polega różnica w użyciu pętli DO…WHILE oraz WHILE? Zgłoś Prowadzącemu zajęcia zakończenie realizacji tego polecenia.

Uruchamianie programów w trybie debugowania

Debugger oferuje funkcję uruchomienia tworzonego programu w trybie pracy krokowej, przejście do pracy ciągłej i z powrotem do pracy krokowej, zatrzymanie warunkowe bądź bezwarunkowe działania programu, sprawdzenie (podgląd) wartości zmiennych, stosu wywołań, stan wyjątków, etc. Głównym elementem Debuggera są punkty zatrzymania działania programu - tzw. Breakpointy. Pod pojęciem Breakpointu należy rozumieć specjalnie oznaczoną przez programistę linijkę kodu, która spowoduje zatrzymanie przed nią wykonywania programu i przejście w tryb pracy krokowej.
W celu uruchomienia tworzonego programu w trybie debugowania należy:

1. Umieścić kursor w linii kodu, w której ma nastąpić zatrzymanie działania programu,
2. Ustawić w tym miejscu Breakpoint – menu **Debug** -> **Toggle Breakpoint**,
3. Uruchomić program – menu **Debug** -> **Start Debugging**,

Uwaga! W celu uruchomienia programu od początku w trybie krokowym bez zaznaczania Breakpointu - menu **Debug** -> **Step Into**

Program zatrzymuje swoje działanie w Breakpoincie i działa w trybie krokowym. Użytkownik w tym trybie ma możliwość między innymi zadania polecenia:

- wykonania kolejnej linii kodu - menu **Debug** -> **Step Over**
- wykonania kolejnej linii kodu z wchodzeniem do wnętrza wszystkich metod/funkcji/procedur (o ile występują w kodzie programu) - menu **Debug** -> **Step In**
- kontunuowania działania programu do kolejnego ustawionego Breakpointu - menu **Debug** -> **Continue**
- całkowitego zatrzymania działania programu - menu **Debug** -> **Stop Debugging**

Polecenie 9:
Napisz program, który będzie wyświetlał na ekranie imię użytkownika programu zdefiniowaną liczbę razy. Program powinien umożliwiać wprowadzenie ciągu znaków reprezentujących imię oraz liczbę wyświetleń. Działanie programu przetestuj w trybie ciągłym (normalnym) oraz w trybie krokowym.

Generator liczb losowych

Przydatnym narzędziem na potrzeby realizacji programów jest wbudowany w język programowania C# generator liczb losowych oferowany za pomocą klasy:

```
Random liczba = new Random();
```

Przypisanie wartości wylosowanej do przykładowej zmiennej liczbowej jest realizowane np. przy pomocy polecenia:

```
a = liczba.Next(min,max);
```

gdzie min oraz max reprezentują granice przedziału losowania, przy czym przedział min-max jest domknięty z lewej oraz otwarty z prawej strony przedziału.

Polecenie 10:
Przetestuj działanie programu, którego listing zapisano poniżej. Sprawdź wpływ zmian położenia granic losowania na uzyskiwane rezultaty działania programu.

```
Random liczba = new Random();
Int32 a, b;

for (b = 0; b <= 20; b++)
{
    a = liczba.Next(10,20);
    Console.WriteLine(a);
};
Console.ReadKey();
```

Polecenie 11:
Wykorzystując poznaną w tym ćwiczeniu wiedzę napisz program, którego działanie polega na odgadnięciu przez użytkownika liczby wylosowanej przez komputer. Losowana liczba mieści się w przedziale 0-100, a użytkownik ma w sumie 6 prób na odgadnięcie liczby. Po nieudanej próbie komputer powinien podpowiedzieć czy wylosowana liczba jest mniejsza/większa od podanej przez użytkownika. Program powinien wyświetlać nr próby oraz wymienione wyżej podpowiedzi. Program powinien zakończyć działanie, jeżeli użytkownik odgadnie liczbę, albo gdy wyczerpie liczbę prób. W konstrukcji programu można wykorzystać instrukcje warunkowe oraz iteracyjne omówione w ćwiczeniu, natomiast nie należy stosować dodatkowych elementów w postaci procedur, funkcji, etykiet, funkcji skoku etc.