Pętle

Pętle (ang. loops), zwane też instrukcjami iteracyjnymi, stanowią podstawę prawie wszystkich algorytmów. Lwia część zadań wykonywanych przez programy komputerowe opiera się w całości lub częściowo właśnie na pętlach.
Pętla to element języka programowania, pozwalający na wielokrotne, kontrolowane wykonywanie wybranego fragmentu kodu. Liczba takich powtórzeń (zwanych cyklami lub iteracjami pętli) jest przy tym ograniczona w zasadzie tylko inwencją i rozsądkiem programisty. Te potężne narzędzia dają więc możliwość zrealizowania niemal każdego algorytmu.
Pętle są też niewątpliwie jednym z atutów C++: ich elastyczność i prostota jest większa niż w wielu innych językach programowania.

Pętla for

Jedną z najczęściej używanych pętli jest pętla `for`. Konstrukcję tej pętli możemy przedstawić następująco:

```
for (inicjacja_warunkow_poczatkowych; wyrazenie_warunkowe; wyrazenie_zwiekszajace)
      {
            blok instrukcji;
      }
```

Poszczególne elementy powyższej konstrukcji omówimy na poniższym przykładzie.

Zadanie 1.
Stworzyć plik `petla.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
int main()
{
   int odp;
   cout << "Podaj liczbe (1-20): ";
   cin >> odp;
   for(int i = 1; i <= odp; i++)
   {
      cout << "i: " << i << endl;
   }
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Zapis `int i = 1` to inicjacja licznika pętli, czyli przypisanie wartości początkowej (w naszym przypadku równej 1) zmiennej całkowitej `i` (można ją zadeklarować również przed pętlą, a w pętli odwołać się do samej zmiennej). Następnie sprawdzane jest wyrażenie warunkowe `i <= odp`. Jeśli jest ono prawdziwe, to wykonany zostanie blok instrukcji (w naszym przypadku wyświetlana będzie aktualna wartość zmiennej `i`), jeśli jest fałszywe, to pętla zakończy działanie. Ostatnia część pętli to zwiększenie (bądź zmniejszenie) wartości licznika. W naszym przypadku licznik jest zwiększany o jeden (zapis `i++`). Następnie ponownie sprawdzamy wyrażenie warunkowe I jeśl jest ono prawdziwe, to zwiększamy liczni o jeden. Czynności te powtarzamy dotąd, aż wyrażenie warunkowe będzie fałszywe. W konsekwencji wynikiem powyższego programu jest wyświetlenie na ekranie liczb od 1 do wczytanej wcześniej liczby przechowywanej w zmiennej `odp`.

Zadanie 2.
Napisać program, który wypisze na ekranie liczby
a) 2, 5, 8, 11, 14,
b) 32, 16, 8, 4, 2.

Zadanie 3.
Napisać program, który oblicza średnią arytmetyczną liczb całkowitych od 1 do liczby podanej przez użytkownika. Wypisać obliczoną średnią na ekranie monitora.

Zadanie 4.
Wczytać dwie liczby całkowite dodatnie a i b, które wyznaczą przedział liczbowy [a,b]. Następnie obliczyć iloczyn liczb całkowitych należących do tego przedziału oraz wypisać go na ekran monitora.

Zadanie 5.
Napisać program wyświetlający na ekranie kolejne liczby całkowite z przedziału [0,a], które są podzielne przez 3 lub przez 4 (Liczbę całkowitą a>0 wczytuje użytkownik z klawiatury).

Zadanie 6.
Napisać program, który pobiera od użytkownika liczbę naturalną n i oblicza silnię tej liczby.

Pętle warunkowe do while i while

Oprócz pętli `for` poznamy dwie kolejne, które zwane są pętlami warunkowymi. Miano to określa całkiem dobrze ich zastosowanie: ciągłe wykonywanie kodu, dopóki spełniony jest określony warunek. Pętla sprawdza go przy każdym swoim cyklu - jeżeli stwierdzi jego fałszywość, natychmiast kończy działanie.
Pierwsza z nich jest pętla `do...while`. Jej konstrukcja wygląda następująco:

```
do
{
      instrukcje;
}
while (warunek);
```

Pętla wykonuje instrukcje tak długo, jak warunek na końcu będzie prawdziwy. Gdy stanie się fałszywy zakończy swoje działanie. Zauważmy, że warunek jest sprawdzany na końcu, a zatem instrukcje wykonają się przynajmniej raz.

Zadanie 1.
Stworzyć plik `dowhile.cpp` i wpisać w nim następujący kod:

```
#include<iostream>
using namespace std;
int main()
{
  int liczba;
  do{
    cout << "Wprowadz liczbe wieksza od 5: ";
    cin >> liczba;
  }
  while (liczba <= 5);
  cout << "W koncu :-)" << endl;
  system("pause");
  return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Myślę, że powyższe zadanie jest na tyle proste, że nie wymaga komentarza.

Ostatnią z omawianych pętli jest pętla `while`. Ma ona postać:

```
while (warunek)
{
      instrukcje;
}
```

Widzimy, że najpierw sprawdzany jest warunek i jeśli jest prawdziwy, wykonywane są instrukcje. Jeśli natomiast jest fałszywy pętla kończy swoje działanie. Może się zatem zdarzyć, że instrukcje w tej pętli nie wykonają się ani razu (w odróżnieniu od pętli `do...while`). Poza tą subtelną różnicą (którą po drobnych poprawkach w kodzie możemy wyeliminować) nie ma znaczenia, czy użyjemy pętli `while` czy `do...while`.

Zadanie 2.
Stworzyć plik `while.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
int main()
{
   int liczba = 59;
   int odp;
   cout << "Sprobuj odgadnac liczbe z przedzialu 1-100: ";
   cin >> odp;
   while (odp != liczba)
   {
      if (odp < liczba)
      {
         cout << "Liczba jest zbyt mala." << endl;
      }
      else
      {
         cout << "Za duza liczba." << endl;
      }
      cout << "Sprobuj jeszcze raz: ";
      cin >> odp;
   }
   cout << "Trafiles(as) Brawo!" << endl;
   system("pause");
   return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Powyższy program, choć wydaje się bardziej skomplikowany od poprzedniego, nie powinien być trudny do zrozumienia.

Ktoś może zapytać zatem, czy jest jakaś różnica między pętlą `for` a pętlami `while` lub `do...while`? W niektórych

sytuacjach rzeczywiście nie ma znaczenia której pętli użyjemy. Zobaczmy to na przykładzie fragmentu kodu poniżej.

```
#include <iostream>
using namespace std;
int main()
{
 cout<<"Rozwiazanie przy uzyciu petli for:"<<endl;
    for(int k=1; k<6; k++)
      {
         cout <<k<< " ";
      }
      cout<<endl;

  cout<<"Rozwiazanie przy uzyciu petli do...while:"<<endl;
     int k=1;
     do
      {
         cout<<k<<" ";
         k++;
       }
     while(k<6);
     cout<<endl;

     int k=1;
cout<<"Rozwiazanie przy uzyciu petli while:"<<endl;
     while(k<6)
       {
        cout <<k<<" ";
        k++;
        }
   cout<<endl;
   system("pause");
   return 0;
}
```

Za każdym razem wypisane zostaną liczby: 1 2 3 4 5. Niemniej często zdarza się tak, że nie chcemy znać od początku ilości cykli (iteracji) pętli (a tak jest w przypadku pętli `for`), tylko decyzję o tym, czy pętla ma się wykonywać dalej czy nie, będziemy podejmować w każdym cyklu. Przeanalizujmy poniższy program.

```
#include <iostream>
using namespace std;
int main()
{
 char litera;
```

```
  do
   {
     cout<<"Podaj znak z klawiatury do momentu napotkania litery k ";
     cin>>litera;
   }
  while(litera!='k');
cout<<"W koncu ja znalazles:)"<<endl;
   system("pause");
   return 0;
}
```

Podobny program to ten z zadania 2. Wczytujemy z klawiatury dowolne znaki, aż do momentu trafienia litery k. Zauważmy, że to my decydujemy jak długo będziemy wklepywać te znaki. Za pomocą pętli `for` trudno byłoby napisać ten program (a na pewno nie w tak prosty sposób jak przy użyciu pętli `do...while`). Problem byłby oczywiście z ilością iteracji (czyli z licznikiem) jaka ma się wykonać.

Używając pętli `while` lub `do...while` napisać poniższe programy.

Zadanie 3.
Napisać program, który oblicza potęgę 2^n, gdzie liczbę naturalną n podaje użytkownik.

Zadanie 4.
Napisać program, który obliczy sumę wyrazów ciągu arytmetycznego postaci: 1,4,7,10,...,31.

Zadanie 5.
Zmodyfikować zadanie 8 z poprzedniego arkusza tak, aby menu z wczytywaniem dwóch liczb rzeczywistych oraz poszczególnymi działaniami arytmetycznymi wykonywanymi na nich pojawiały się na ekranie ponownie (po wykonaniu wcześniej wybranego działania) do momentu, aż nie wybierzemy opcji oznaczającej wyjście z programu. Więcej szczegółów podam na zajęciach.