Instrukcje sterujące: pętle

Pętla to jedna z trzech podstawowych konstrukcji programowania strukturalnego (obok instrukcji warunkowej i instrukcji wyboru). Umożliwia cykliczne wykonywanie ciągu instrukcji określoną liczbę razy lub do momentu zajścia pewnych warunków.

Pętle wykorzystujemy zazwyczaj, gdy chcemy wielokrotnie wykonać ten sam ciąg instrukcji. Dzięki ich zastosowaniu nie musimy wiele razy przepisywać tego samego kodu.

W języku C wyróżniamy 2 podstawowe rodzaje pętli:

- **Pętle warunkowe (repetycyjne)** – konstrukcja pozwalająca na wykonywanie pewnych instrukcji aż do odpowiedniej zmiany warunków. Przeważnie wyrażenie testujące sprawdzane jest na początku lub na końcu pętli. Przykładami są pętle: **while** oraz **do ... while**.
- **Pętle licznikowe (iteracyjne)** - konstrukcja pozwalająca na wykonywanie pewnych instrukcji określoną liczbę razy. W najprostszym przypadku na początku pętli specjalna zmienna sterująca (licznikowa) jest ustawiana na wartość początkową, następnie przy każdym obiegu pętli jej wartość jest zwiększana o jeden, aż do osiągnięcia górnego limitu. Często pętla taka może odliczać w dół lub zmienna może być modyfikowana o wartość inną niż 1. Przykładem jest pętla **for**.

Pętla *while*

Postać ogólna pętli *while* wygląda następująco:

```
while (warunek) {
  /* instrukcje do wykonania w pętli */
}
/* dalsze instrukcje */
```

Przykład

```
#include<stdio.h>
int main()
{
    int i = 1;
    while(i <= 10) // warunek petli
    {
        printf("i= %d\n", i);
        i++; // i = i+1;
    }
    return 0;
}
```

Rozważmy powyższy przykład:

Program ma za zadanie wypisać na ekranie bieżącą wartość zmiennej *i*. Na początku (przy deklaracji) wartość zmiennej *i* wynosi 1. Następnie mamy pętlę, której warunek mówi: „wykonuj instrukcje wewnątrz pętli dopóki *i<=10*”. Zanim przejdziemy do instrukcji *printf* musimy sprawdzić, czy warunek pętli jest prawdziwy. Nasze *i* jest równe 1, zatem 1<=10 – prawda. Więc program wypisze komunikat: „i = 1”. I następnie nastąpi inkrementacja zmiennej *i* o 1.

Później działanie pętli się ni kończy. Ponownie sprawdzamy warunek *i<=10*. Nasze *i* tym razem wynosi 2 (konsekwencja inkrementacji). 2<=10, zatem wypisany zostanie komunikat: „i=2”. I ponownie przejdziemy do sprawdzenia warunku pętli.

W sumie pętla wykona się 10 razy wypisując na ekran wartości *i* od 1 do 10.

Zauważyć należy również, że instrukcja inkrementacji jest tutaj niezbędna. Gdyby jej nie było, to wartość zmiennej *i* nie uległaby zmianie, a co za tym idzie – pętla nigdy by się nie zakończyła, bo 1<=10 jest zawsze prawdziwe.

ZADANIE

Przetestuj działanie pętli *while*.

Pętla *do … while*

Postać ogólna pętli *do … while* wygląda następująco:

```
do {
  /* instrukcje do wykonania w pętli */
} while (warunek);
/* dalsze instrukcje */
```

Przykład

```
#include<stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("i= %d\n", i);
        i++; // i = i+1;
    } while(i <= 10); // warunek petli
    return 0;
}
```

Rozważmy powyższy przykład:

Program ma za zadanie wypisać na ekranie bieżącą wartość zmiennej *i*. Na początku (przy deklaracji) wartość zmiennej *i* wynosi 1. Następnie mamy pętlę, w której najpierw zostanie wypisana na ekran wartość *i* oraz dokona się inkrementacja tej zmiennej, a dopiero później sprawdzony zostanie warunek mówiący: „sprawdź czy *i<=10*”. Zatem efektem pierwszego obiegu pętli będzie wypisanie przez program komunikatu: „i = 1” oraz zwiększenie wartości *i* o 1 (*i*=2).

Dalej zostanie sprawdzony warunek czy i<=10. Nasze i=2, zatem 2<=10 – prawda. Więc program ‘przejdzie’ ponownie do wykonania instrukcji zawartych w bloku *do{ }*, a potem ponownie sprawdzany będzie warunek pętli.

Uwaga techniczna: Zauważ, że na końcu linii zawierającej warunek pętli znajduje się średnik.

ZADANIE

Przetestuj działanie pętli *do … while*.

Różnice między *while* a *do … while*

Na pierwszy rzut oka w obydwu powyższych przykładach dzieje się to samo – przecież wyniki wypisywane przez powyższe programy są takie same. A jednak sposób działania każdej z pętli jest inny.

Zasadniczą cechą pętli *do … while* różniącą ją od pętli *while* jest to, że w pętli *do … while* najpierw wykonywane są instrukcje wewnątrz pętli, a dopiero później sprawdzany jest warunek pętli. Zatem instrukcje pętli *do … while* zostaną wykonane co najmniej 1 raz. Jest to bardzo ważna różnica i w wielu przypadkach wręcz decydująca o dalszym działaniu programu.

Dlatego jeszcze raz szczegółowo przeanalizujmy obydwa programy pod względem wartości zmiennej *i* oraz wypisywanych komunikatów, a także kolejności wykonywania poszczególnych akcji.

```c
#include<stdio.h>
int main()
{
    int i = 1;
    while(i <= 10) // warunek petli
    {
        printf("i= %d\n", i);
        i++; // i = i+1;
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("i= %d\n", i);
        i++; // i = i+1;
    } while(i <= 10); // warunek petli
    return 0;
}
```

Numer obiegu pętli	while		do … while	
	Akcja	Wynik	Akcja	Wynik
1	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 1<=10. „i=1” i=2	Wypisz i. Zwiększ i. Czy i<=10?	„i=1” i=2 Tak, bo 2<=10.
2	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 2<=10. „i=2” i=3	Wypisz i. Zwiększ i. Czy i<=10?	„i=2” i=3 Tak, bo 3<=10.
3	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 3<=10. „i=3” i=4	Wypisz i. Zwiększ i. Czy i<=10?	„i=3” i=4 Tak, bo 4<=10.
4	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 4<=10. „i=4” i=5	Wypisz i. Zwiększ i. Czy i<=10?	„i=4” i=5 Tak, bo 5<=10.
5	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 5<=10. „i=5” i=6	Wypisz i. Zwiększ i. Czy i<=10?	„i=5” i=6 Tak, bo 6<=10.
6	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 6<=10. „i=6” i=7	Wypisz i. Zwiększ i. Czy i<=10?	„i=6” i=7 Tak, bo 72<=10.
7	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 7<=10. „i=7” i=8	Wypisz i. Zwiększ i. Czy i<=10?	„i=7” i=8 Tak, bo 8<=10.
8	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 8<=10. „i=8” i=9	Wypisz i. Zwiększ i. Czy i<=10?	„i=8” i=9 Tak, bo 9<=10.
9	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 9<=10. „i=9” i=10	Wypisz i. Zwiększ i. Czy i<=10?	„i=9” i=10 Tak, bo 10<=10.
10	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 10<=10. „i=10” i=11	Wypisz i. Zwiększ i. Czy i<=10?	„i=10” i=11 Nie, bo 11>10.
11	Czy i<=10?	Nie, bo 11>10.		

ZADANIE

Sprawdź działanie obydwu analizowanych programów (z *while* i *do…while*) przy zmienionym warunku pętli na *i<=0*. Czy obydwa programy wypisują te same wyniki? Dlaczego tak się dzieje?

Przykłady

```c
#include<stdio.h>
int main()
{
    int i = 10;
    while(i >= 1)
    {
        printf("i= %d\n", i);
        i--;
    }
    printf("po zakonczeniu petli i= %d\n", i);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 10;
    do
    {
        printf("i= %d\n", i);
        i--;
    } while(i >= 1);
    printf("po zakonczeniu petli i= %d\n", i);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 10;
    while(i < 5)
        i++;
    printf("po zakonczeniu petli i= %d\n", i);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 10;
    do
        i++;
    while(i < 5);
    printf("po zakonczeniu petli i= %d\n", i);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 100;
    int j = 0;
    while(i > 0 && j < 100)
    {
        printf("i= %d; \tj= %d\n", i, j);
        i--;
        j = j+2;
    }
    printf("po zakonczeniu petli i= %d; \tj= %d\n", i, j);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 100;
    int j = 0;
    do
    {
        printf("i= %d; \tj= %d\n", i, j);
        i--;
        j = j+2;
    }
    while(i > 0 && j < 100);
    printf("po zakonczeniu petli i= %d; \tj= %d\n", i, j);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    while(0 >= 0)
        printf("never ending loop ...\n");
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    do
        printf("never ending loop ...\n");
    while(0 <= 0);
    return 0;
}
```

ZADANIE

Sprawdź działanie powyższych przykładów. Zastanów się nad sposobem działania każdego przykładu analizując akcje dokonujące się w kodzie (np. przypisywanie wartości do zmiennych, zmiana ich wartości, wypisywanie komunikatów itp.).

Pętla *for*

Pętla for jest chyba najbardziej popularną pętlą. Jest ona podobna w działaniu do pętli *while*, z tym że umożliwia wpisanie ustawiania zmiennej sterującej, sprawdzania warunku pętli i inkrementowania zmiennej sterującej w jednej linijce, co często zwiększa czytelność kodu.

Postać ogólna pętli *for* wygląda następująco:

```c
for (wyrażenie1; wyrażenie2; wyrażenie3) {
  /* instrukcje do wykonania w pętli */
}
/* dalsze instrukcje */
```

gdzie:

- wyrażenie1 - instrukcja, która będzie wykonana przed pierwszym obiegiem pętli. Zwykle jest to inicjalizacja zmiennej, która będzie służyła jako „licznik" obiegów pętli;
- wyrażenie2 - warunek trwania pętli. Pętla wykonuje się tak długo, jak prawdziwy jest ten warunek;
- wyrażenie3 - instrukcja, która wykonywana będzie po każdym przejściu pętli (także po ostatnim). Zamieszczone są tu instrukcje, które zwiększają licznik o odpowiednią wartość.

Przykład

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=1; i<=10; i++)
    {
        printf("i= %d\n", i);
    }
    return 0;
}
```

Powyższy przykład realizuje zadanie wyświetlenia na ekranie bieżącej wartości zmiennej *i*. Wizualny jego efekt działania będzie dokładnie taki sam, jak w przypadku analizowanych programów z pętlą *while* i *do…while*, tzn. wypisanie wartości od 1 do 10. Natomiast akcje dokonujące się wewnątrz programu będą nieco inne.

Aby lepiej zrozumieć działanie pętli *for* dokonajmy analizy powyższego przykładu pętli *for* w odniesieniu do pętli *while*:

Numer obiegu pętli	while		for	
	Akcja	Wynik	Akcja	Wynik
1	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 1<=10. „i=1" i=2	Przypisz i wart. 1. Czy i<=10? Wypisz i. Zwiększ i.	i=1 Tak, bo 1<=10. „i=1" i=2.
2	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 2<=10. „i=2" i=3	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 2<=10. „i=2" i=3
3	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 3<=10. „i=3" i=4	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 3<=10. „i=3" i=4
4	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 4<=10. „i=4" i=5	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 4<=10. „i=4" i=5
5	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 5<=10. „i=5" i=6	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 5<=10. „i=5" i=6
6	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 6<=10. „i=6" i=7	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 6<=10. „i=6" i=7
7	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 7<=10. „i=7" i=8	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 7<=10. „i=7" i=8
8	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 8<=10. „i=8" i=9	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 8<=10. „i=8" i=9
9	Czy i<=10? Wypisz i.	Tak, bo 9<=10. „i=9"	Czy i<=10? Wypisz i.	Tak, bo 9<=10. „i=9"

	Zwiększ i.	i=10	Zwiększ i.	i=10
10	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 10<=10. „i=10” i=11	Czy i<=10? Wypisz i. Zwiększ i.	Tak, bo 10<=10. „i=10” i=11
11	Czy i<=10?	Nie, bo 11>10.	Czy i<=10?	Nie, bo 11>10.

Jeszcze raz spójrzmy na programy z *while* i *for*, wypisujące liczby z zakresu 1-10:

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=1; i<=10; i++)
    {
        printf("i= %d\n", i);
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i = 1;
    while(i <= 10) // warunek petli
    {
        printf("i= %d\n", i);
        i++; // i = i+1;
    }
    return 0;
}
```

Dla powyższych przykładów możemy określić następującą równość:

```c
for (wyrażenie1; wyrażenie2; wyrażenie3) {
  /* instrukcje do wykonania w pętli */
}
/* dalsze instrukcje */
```

=

```c
{
  wyrażenie1;
  while (wyrażenie2) {
    /* instrukcje do wykonania w pętli */
    wyrażenie3;
  }
}
/* dalsze instrukcje */
```

Spójrzmy na inne przykłady użycia pętli *for*.

Przykłady

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=0; i<=1000; i=i+100)
    {
        printf("i= %d\n", i);
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=1000; i>=0; i=i-100)
    {
        printf("i= %d\n", i);
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i=0, j=100;
    for(i; i<j; i=i+20)
    {
        printf("i= %d\n", i);
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i=0, j=10;
    for(i; i<=10 && j>=0; i++)
    {
        printf("i= %d; \tj= %d\n", i, j);
        j--;
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i, j;
    for(i=0; i<=5; i++)
        for(j=0; j<=5; j++)
            printf("i= %d; \tj= %d\n", i, j);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=0; i>=0; i++)
        printf("i= %d\n", i);
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższych przykładów ze szczegółową analizą akcji wykonywanych przez program.

Instrukcje *break* i *continue*

Instrukcja ***break*** pozwala na opuszczenie wykonywania pętli w dowolnym momencie.

W przeciwieństwie do *break*, która przerywa wykonywanie pętli, instrukcja ***continue*** powoduje przejście do następnej iteracji, o ile tylko warunek pętli jest spełniony.

Przykłady

```c
#include<stdio.h>
int main()
{
    int i=0;
    while(i>=0)
    {
        printf("i= %d\n", i);
        if(i>=20)
            break;
        i++;
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i;
    // przyklad petli nieskonczonej
    for(i=0 ; ; i++) // brak warunku trwania petli
    {
        printf("i= %d\n", i);
        if(i>=10)
            break;
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i=0;
    while(i < 8)
    {
        i++;
        if(i == 2 || i==5 || i==6)
            continue;
        printf("i= %d\n", i);
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int i;
    for(i=0; i<=10; i++)
    {
        if(i >=2 && i <=8)
            continue;
        printf("i= %d\n", i);
    }
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższych przykładów ze szczegółową analizą akcji wykonywanych przez program.

Instrukcja *break* znajduje zastosowanie m.in. w sytuacji, gdy chcemy odnaleźć jakąś wartość w ciągu, która jako pierwsza spełnia określone warunki, np.:

- znalezienie najmniejszej liczby z przedziału 0-1000, która jest podzielna jednocześnie przez 3, 4, 5 i 7;
- zakończenie działania pętli (lub nawet programu) przy podaniu przez użytkownika określonego znaku;

Instrukcja *continue* z kolei znajduje zastosowanie m.in. w sytuacji gdy chcemy wypisać wszystkie elementy ciągu poza tymi, które spełniają określone warunki, np.:

- wyświetlenie liczb z zakresu 0-100 z pominięciem tych, które są podzielne przez 5;

Zadania

Poniższe zadania rozwiąż z wykorzystaniem pętli.

1. Napisz program, który wyświetli wszystkie liczby parzyste (nieparzyste) z zakresu 1-100.
2. Napisz program, który wyświetli wszystkie liczby z przedziału 0-1000, które są podzielne jednocześnie przez 3, 4, 5 i 7.
3. Napisz program, który wyświetli wszystkie liczby pierwsze z zakresu 1-100.
4. Napisz program, który wyświetli wszystkie liczby z zakresu 1-100 podzielne przez 5.
5. Napisz program umożliwiający wypisanie litery E złożonej z liter E. Użytkownik podaje wysokość i szerokość litery.
   Wskazówka: zauważ, że aby wyświetlony ciąg znaków miał postać litery E, to minimalna liczba wierszy (wysokość litery) powinna wynosić co najmniej 5, zaś liczba kolumn (szerokość litery) – co najmniej 3: