



Fundusze
Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska

Unia Europejska
Europejski Fundusz Społeczny



Politechnika  Białostocka

Wydział Elektryczny

Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja do pracowni specjalistycznej

Temat ćwiczenia:

**JĘZYK C - INSTRUKCJE ITERACYJNE FOR,
WHILE I DO...WHILE, INSTRUKCJE
CONTINUE, BREAK I GOTO**

Ćwiczenie nr INF_D04

Pracownia specjalistyczna z przedmiotu:

Informatyka

Kod: **EDS1B 1007**

Opracował:

dr inż. Jarosław Forenc

Białystok 2018

Materiały zostały opracowane w ramach projektu „PB2020 – Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Spis treści

1. Opis stanowiska	3
1.1. Stosowana aparatura	3
1.2. Oprogramowanie.....	3
2. Wiadomości teoretyczne.....	3
2.1. Instrukcja for.....	3
2.2. Operatory zwiększania (++) i zmniejszania (--).	7
2.3. Dodatkowe uwagi do instrukcji for	8
2.4. Zagnieżdżanie pętli for	10
2.5. Instrukcja kontynuacji (continue)	12
2.6. Instrukcja break.....	14
2.7. Instrukcja skoku (goto)	14
2.8. Instrukcja while.....	15
2.9. Instrukcja do...while.....	21
3. Przebieg ćwiczenia.....	23
4. Literatura.....	26
5. Pytania kontrolne	26
6. Wymagania BHP	27

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

© Wydział Elektryczny, Politechnika Białostocka, 2018 (wersja 1.0)

Wszelkie prawa zastrzeżone. Żadna część tej publikacji nie może być kopiowana i odtwarzana w jakiegokolwiek formie i przy użyciu jakichkolwiek środków bez zgody posiadacza praw autorskich.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows 10.

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub nowsze zawierające kompilator Microsoft Visual C++.

2. Wiadomości teoretyczne

2.1. Instrukcja for

W programach komputerowych bardzo często pewien fragment programu wykonywany jest wielokrotnie lub dla zmiennych przyjmujących kolejne wartości. W takich przypadkach stosowane są pętle czyli instrukcje iteracyjne. Podstawową instrukcją iteracyjną jest pętla **for**. W poniższym programie zastosowano pętlę **for** do pięciokrotnego wyświetlenia tego samego tekstu.

Program wyświetlający pięć razy ten sam tekst.

```
#include <stdio.h>

int main(void)
{
    int i;

    for (i = 0; i < 5; i = i + 1)
        printf("Programowanie nie jest trudne\n");

    return 0;
}
```

Wynikiem działania programu będzie następujący wydruk:

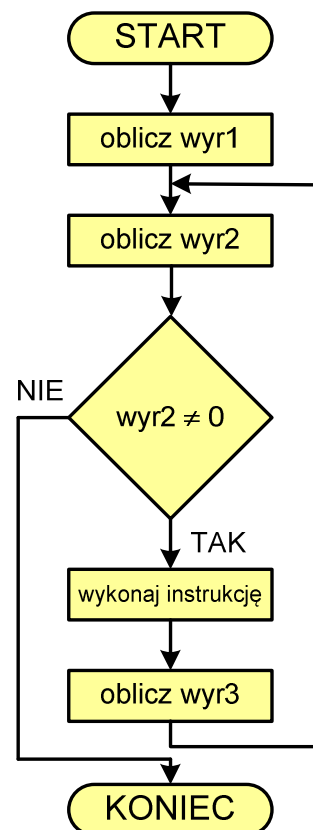
```
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
```

Ogólna postać instrukcji **for** jest następująca:

```
for (wyr1; wyr2; wyr3)
    instrukcja;
```

Instrukcja **for** tworzy pętlę pracującą w następujący sposób:

1. Wyliczane jest **wyr1** (jeśli występuje).
2. Wyliczane jest **wyr2**. Jeśli jego wartość jest różna od zera (czyli jest ono prawdziwe), to następuje przejście do kroku 3. W przeciwnym przypadku instrukcja **for** kończy działanie.
3. Wykonywana jest **instrukcja**.
4. Wyliczana jest wartość **wyr3**. Działanie jest wznowiane od kroku 2, czyli obliczania **wyr2**.



W pierwszym programie zmienna występująca w pętli **for** (**i**) pełniła tylko rolę tzw. licznika pętli. W kolejnym programie zmienna ta jest wykorzystywana także w obliczeniach.

Poniższy program zamienia temperaturę ze skali Fahrenheita na temperaturę w skali Celsjusza. W jednym programie zamieniane są wartości od 0 F do 50 F co 10 F.

Program przeliczający temperaturę ze skali Fahrenheita na skalę Celsjusza.

```
#include <stdio.h>
```

```
int main(void)
{
    int f;

    for (f = 0; f < 60; f = f + 10)
        printf("%2d F to %6.2f C\n", f, 5*(f-32.0)/9);

    return 0;
}
```

Wynikiem działania programu jest następujący wydruk:

```
 0 F to -17.78 C
10 F to -12.22 C
20 F to  -6.67 C
30 F to  -1.11 C
40 F to   4.44 C
50 F to  10.00 C
```

W nawiasach instrukcji **for** występują trzy części (wyrażenia) oddzielone od siebie średnikami:

- pierwsza część inicjalizuje pętlę (**f = 0**), jest ona wykonywana tylko raz, przed wejściem do pętli;
- druga część jest warunkiem sterującym powtarzaniem pętli (**f < 60**). Jeśli jest on prawdziwy to wykonywana jest instrukcja **printf()** znajdująca się w kolejnym wierszu kodu programu;
- następnie, po wykonaniu instrukcji **printf()**, wykonywana jest trzecia część, w której zwiększana jest wartość zmiennej wykorzystywanej w pętli (**f = f + 10**) i ponownie sprawdzany jest warunek sterujący powtarzaniem pętli (**f < 60**).

Zmienne występujące w pętlach nazywane są kolejnymi literami alfabetu: **i**, **j**, **k**, **l**, ..., chyba, że z kontekstu programu wynika użycie zmiennej o innej nazwie (tak jak w przypadku programu zamieniającego temperatury: **f** - zmienna przechowująca temperaturę w skali Fahrenheita). Zmienne te mogą przyjmować dowolne wartości, zależnie od trzech wyrażen w nawiasach pętli **for**. Poniżej podano przykładowe pętle, w których instrukcja **printf()** wyświetla kolejne wartości przyjmowane przez zmienną całkowitą **i**.

```
for (i = 0; i < 10; i = i + 1)
    printf("%d ", i);
```

0 1 2 3 4 5 6 7 8 9

```
for (i = 1; i < 10; i = i + 2)
    printf("%d ", i);
```

1 3 5 7 9

```
for (i = 10; i > 0; i = i - 1)
    printf("%d ", i);
```

10 9 8 7 6 5 4 3 2 1

```
for (i = -9; i <= 9; i = i + 3)
    printf("%d ", i);
```

-9 -6 -3 0 3 6 9

Jeśli w każdej iteracji wartość zmiennej sterującej pętli jest zwiększana lub zmniejszana o **jeden** to zamiast zapisów:

```
for (i = 0; i < 10; i = i + 1)
    instrukcja;
```

```
for (i = 10; i > 0; i = i - 1)
    instrukcja;
```

stosuje się operatory zwiększania (++) i zmniejszania (--), nazywane także operatorami inkrementacji i dekrementacji:

```

for (i = 0; i < 10; i++)
    instrukcja;

for (i = 10; i > 0; i--)
    instrukcja;

```

Operatory te zostały opisane w kolejnym rozdziale instrukcji.

2.2. Operatory zwiększania (++) i zmniejszania (--)

Operator ++ służy do zwiększania wartości zmiennej o 1, zaś operator -- służy do zmniejszania wartości zmiennej o 1. Operatory te są jednoargumentowe. Można stosować je tylko do zmiennych (nie można ich stosować do wyrażeń). Operatory te mogą występować jako przedrostek lub przyrostek (Tabela 1).

Tabela 1. Operatory ++ i --

Zapis	Operator	Znaczenie
++i	preinkrementacji	Operator występuje przed nazwą zmiennej (i), wartość zmiennej modyfikowana jest przed jej użyciem.
--i	predekrementacji	
i++	postinkrementacji	Operator występuje po nazwie zmiennej (i), wartość zmiennej modyfikowana jest po użyciu jej poprzedniej wartości.
i--	postdekrementacji	

Rozpatrzmy następujący fragment programu:

```

int i = 2, j;

j = 2 * ++i;
printf("%d %d", i, j);

```

W powyższym programie najpierw wartość zmiennej *i* jest zwiększana o jeden (z 2 do 3). Następnie wykonywana jest operacja mnożenia (2 * 3). Wynik tej operacji (6) przypisywany jest zmiennej *j*. Zatem instrukcja **printf()** wyświetli wartości: **3 6**.

Jeśli operator preinkrementacji zastąpimy operatorem postinkrementacji:

```
int i = 2, j;

j = 2 * i++;
printf("%d %d", i, j);
```

to najpierw wykonywana jest operacja mnożenia ($2 * i = 2 * 2$). Wynik tej operacji (4) przypisywany jest zmiennej *j*. Następnie zmienna *i* zwiększana jest o jeden (z 2 do 3). Instrukcja **printf()** wyświetli zatem: **3 4**.

Miejsce umieszczenia operatorów inkrementacji i dekrementacji nie ma znaczenia w przypadku instrukcji typu:

```
i++;
++i;      równoważne:      i = i + 1;

i--;
--i;      równoważne:      i = i - 1;
```

gdyż efekt końcowy będzie taki sam (zwiększenie lub zmniejszenie wartości zmiennej *i* o 1).

Nie jest zalecane stosowanie operatorów **++** i **--** do zmiennej, która pojawia się w wyrażeniu więcej niż jeden raz. Wynik poniższej instrukcji:

```
i = i++;
```

jest według standardu języka C **niezdefiniowany**.

2.3. Dodatkowe uwagi do instrukcji **for**

Każde z trzech wyrażen w nawiasach pętli **for** jest opcjonalne (może jego nie być), ale nawiasy i średniki są obowiązkowe.

```
for (wyr1; wyr2; wyr3)
    instrukcja;
```


Jeśli nie jest podane **wyr2**, to przyjmuje się, że jest ono prawdziwe. Zatem w prosty sposób można skonstruować pętlę nieskończoną.

```
for ( ; ; )  
    instrukcja;
```

Jeśli w pętli **for** ma być wykonana więcej niż jedna instrukcja, to należy zastosować **instrukcję złożoną**, czyli objąć wszystkie te instrukcje nawiasami klamrowymi. W poniższym programie w pętli **for** wykonywane są trzy instrukcje.

Program obliczający średnią arytmetyczną 6 liczb całkowitych.

```
#include <stdio.h>  
#pragma warning(disable:4996)  
  
int main(void)  
{  
    int    i, x;  
    float  suma = 0.0f;  
  
    for (i = 0; i < 6; i++)  
    {  
        printf("Podaj liczbe nr %d: ", i+1);  
        scanf("%d", &x);  
        suma = suma + x;  
    }  
    printf("Srednia: %.3f\n", suma/6);  
  
    return 0;  
}
```

Przykładowy wynik działania programu:

```
Podaj liczbe nr 1: 3  
Podaj liczbe nr 2: 8  
Podaj liczbe nr 3: 5  
Podaj liczbe nr 4: 7  
Podaj liczbe nr 5: 3  
Podaj liczbe nr 6: 2  
Srednia: 4.667
```

Po nawiasie zamykającym pętli **for** nie stawia się średnika. Konstrukcja ze średnikiem na końcu jest poprawna składniowo (kompilator nie zasygnalizuje błędu), ale oznacza wielokrotne wykonanie **instrukcji pustej** (w poniższym przykładzie - 10 razy). Natomiast **instrukcja** zostanie wykonana tylko jeden raz.

```
for (i = 0; i < 10; i++);  
    instrukcja;
```

Często popełnianym błędem przez początkujących programistów jest wprowadzenie przecinków zamiast średników.

```
for (i = 0, i < 10, i++)  
    instrukcja;
```

W takiej sytuacji kompilator wyświetli błąd:

```
1>c:\prog.cpp() : error C2143: syntax error : missing ';' before ')'
1>c:\prog.cpp() : error C2143: syntax error : missing ';' before ')'
```

Jeszcze innym rodzajem błędu jest podanie niewłaściwego warunku kontynuacji pętli. W poniższym przykładzie funkcja **printf()** nie wykona się ani razu.

```
for (i = 0; i > 10; i++)  
    printf("%d ", i);
```

W następnym przykładzie otrzymujemy pętlę nieskończoną, gdyż warunek **i > 0** jest zawsze prawdziwy:

```
for (i = 1; i > 0; i++)  
    printf("%d ", i);
```

2.4. Zagnieżdżanie pętli for

Zagnieżdżanie pętli **for** polega na tym, że jako instrukcja w pętli występuje kolejna pętla **for**. Pierwsza pętla nazywana jest pętlą **zewnętrzną**, zaś druga -

wewnętrzna. Zasada działania zagnieżdżonych pętli zostanie pokazana na przykładzie poniższego kodu programu.

```
for (i = 1; i <= 3; i++)
    for (j = 1; j <= 2; j++)
        printf("i = %d, j = %d\n", i, j);
```

Wynikiem działania zagnieżdżonych pętli jest wyświetlenie tekstu:

```
i = 1, j = 1
i = 1, j = 2
i = 2, j = 1
i = 2, j = 2
i = 3, j = 1
i = 3, j = 2
```

W zewnętrznej pętli zmienna *i* otrzymuje wartość **1**. Następnie wykonywana jest pętla wewnętrzna, w której zmienna *j* przyjmuje wartości **1** i **2**. Po zakończeniu pętli wewnętrznej następuje powrót do pętli zewnętrznej - zmienna *i* jest zwiększana o **1**, przyjmując wartość **2**. Ponownie wykonywana jest pętla wewnętrzna, itd.

W kolejnym programie zagnieżdżanie pętli zostało wykorzystane do wyświetlenia na ekranie tabliczki mnożenia.

Program wyświetlający tabliczkę mnożenia.

```
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i < 11; i++)
    {
        for (j=1; j<11; j++)
            printf("%2d  ", i*j);
        printf("\n");
    }

    return 0;
}
```

Wynik działania powyższego programu:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
0	20	30	40	50	60	70	80	90	100

Zewnętrzna pętla **for** wykonywana jest 10 razy (i zmienia się od 1 do 10). Wewnętrzna pętla **for** wykonywana jest także 10 razy (j zmienia się od 1 do 10), zatem instrukcja **printf()** będzie wykonana 100 razy.

2.5. Instrukcja kontynuacji (continue)

Instrukcja kontynuacji używana jest wewnątrz pętli **for** i ma postać:

```
continue;
```

Instrukcja ta powoduje bezwarunkowe przejście na początek pętli (do następnej iteracji) z pominięciem instrukcji znajdujących się poniżej **continue**. Dokładniej mówiąc, powoduje przejście do wyliczania wartości **wyr3** w pętli **for**.

Program obliczający średnią arytmetyczną 10 liczb całkowitych wprowadzonych z klawiatury przy uwzględnieniu tylko liczb nieujemnych.

```
#include <stdio.h>
#pragma warning(disable:4996)

int main(void)
{
    int    i, x, ilosc = 0;
    float  suma = 0.0f;

    for (i = 0; i < 10; i++)
    {
        printf("Podaj liczbe nr %d: ", i+1);
        scanf("%d", &x);
    }
}
```

```

        if (x < 0)
            continue;
        suma = suma + x;
        ilosc++;
    }

    if (ilosc > 0)
    {
        printf("Ilosc liczb:   %d\n", ilosc);
        printf("Suma liczb:    %g\n", suma);
        printf("Srednia:        %.3f\n", suma/ilosc);
    }
    else
        printf("Wszystkie liczby sa ujemne\n");

    return 0;
}

```

Przykładowy wynik działania programu:

```

Podaj liczbe nr 1: 3
Podaj liczbe nr 2: -4
Podaj liczbe nr 3: 2
Podaj liczbe nr 4: -4
Podaj liczbe nr 5: 0
Podaj liczbe nr 6: 2
Podaj liczbe nr 7: 1
Podaj liczbe nr 8: -3
Podaj liczbe nr 9: 9
Podaj liczbe nr 10: 2
Ilosc liczb:    7
Suma liczb:     19
Srednia:        2.714

```

W powyższym programie jeśli wartość kolejnej liczby **x** jest mniejsza od zera to wywoływana jest instrukcja **continue**. Powoduje ona pominięcie dwóch instrukcji z bieżącej iteracji:

```

suma = suma + x;
ilosc++;

```

i przejście do kolejnej iteracji (wprowadzania kolejnej liczby).

2.6. Instrukcja break

Instrukcja **break** umożliwia wcześniejsze zakończenie pętli **for**. Wywołanie tej instrukcji powoduje natychmiastowe opuszczenie pętli i przejście do instrukcji znajdującej się bezpośrednio po pętli **for**.

```
for (i = 1; i < 10; i++)
{
    if (i % 5 == 0)
        break;
    printf("%d\n", i);
}
printf("Koniec, i = %d\n", i);
```

Gdy warunek w instrukcji **if** będzie prawdziwy zostanie wykonana instrukcja **break** powodująca przerwanie pętli **for**:

```
1
2
3
4
Koniec, i = 5
```

W przypadku zagnieżdżonych pętli przerywane jest działanie tylko jednej pętli - najbardziej wewnętrznej.

2.7. Instrukcja skoku (goto)

Instrukcja ta w postaci:

```
goto label;
```

przekazuje sterowanie do miejsca w programie oznaczonego etykietą o nazwie **label**. Etykietą można oznaczać tylko całe instrukcje. Skok może odbywać się tylko w ramach tej samej funkcji.

Wystąpienie w poniższym fragmencie programu instrukcji skoku powoduje przeniesienie sterowania do instrukcji rozpoczynającej się od etykiety **dalej**.

```

for (i = 1; i < 10; i++)
{
    if (i % 5 == 0)
        goto dalej;
    printf("%d\n", i);
}

dalej: printf("Koniec, i = %d\n", i);

```

Wynik działania powyższego fragmentu programu:

```

1
2
3
4
Koniec, i = 5

```

W programach w języku C **nie zaleca się** stosowania instrukcji **goto**. Jedynym przypadkiem, kiedy dopuszcza się zastosowanie **goto**, jest wyjście z układu wielu zagnieżdżonych pętli.

2.8. Instrukcja while

Instrukcja **while** służy do tworzenia pętli w języku C.

Składnia tej instrukcji jest następująca:

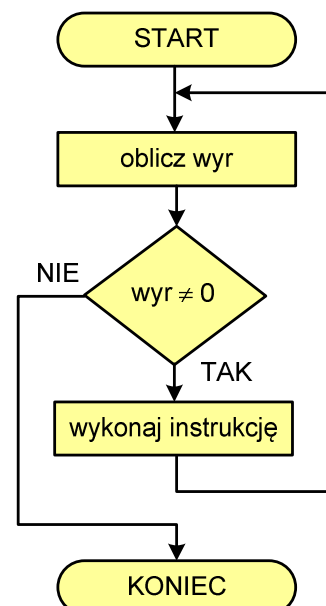
```

while (wyr)
    instrukcja;

```

Zasada działania:

1. Obliczana jest wartość **wyr**.
2. Jeśli wartość **wyr** jest różna od zera, czyli wyrażenie jest prawdziwe, to wykonywana jest **instrukcja**. Następnie wracamy do pkt. 1 (obliczanie wartości **wyr**). Jeśli wartość **wyr** jest równa zero (nie jest ono prawdziwe), to pętla **while** kończy działanie.



Działanie pętli **while** można opisać także w następujący sposób: „dopóki wyrażenie w nawiasach jest prawdziwe wykonuj instrukcję”.

Jako **wyr** najczęściej stosowane jest wyrażenie logiczne. Należy umieścić je w nawiasach zwykłych. Poprawne zakończenie pętli **while** wymaga, aby w jej wnętrzu następowały zmiany (np. modyfikacje wartości zmiennych) wpływające na wartość wyrażenia w nawiasach (**wyr**). W poniższym przykładzie pętla **while** wykonuje się do osiągnięcia przez zmienną **i** wartości **10**.

```
int i = 0;
while (i < 10)
    i++;
```

Jako **instrukcja** po pętli **while** może wystąpić **instrukcja złożona**, czyli kilka instrukcji ograniczonych nawiasami klamrowymi: { i }.

Program sprawdzający ile kolejnych liczb całkowitych należy dodać do siebie, aby otrzymać największą wartość nie przekraczającą **max**.

```
#include <stdio.h>
#pragma warning(disable:4996)

int main(void)
{
    int suma = 0, n = 0;
    int max;

    printf("Podaj max:    ");
    scanf("%d", &max);

    while (suma < max)
    {
        n++;
        suma = suma + n;
    }

    suma = suma - n;
    n--;

    printf("Ilość liczb:    %d\n", n);
    printf("Suma liczb:     %d\n", suma);
```



```
    return 0;
}
```

Przykładowy wynik uruchomienia programu:

```
Podaj max:      100
Ilość liczb:    13
Suma liczb:     91
```

W każdej iteracji zwiększana jest wartość zmiennej **n** o **jeden** (**n++**). Następnie zwiększone **n** dodawane jest do ogólnej sumy (**suma = suma + n;**). Operacje te powtarzane są do momentu, aż **suma** przekroczy wartość **max** - wtedy warunek w pętli **while** nie będzie prawdziwy. Ponieważ suma nie powinna przekroczyć wartości **max**, to należy cofnąć się o jeden krok. Wymaga to wykonania dwóch instrukcji: **suma = suma - n; n--;**.

Pętla **while** występująca w powyższym programie może być zapisana w skróconej postaci:

```
while (suma < max)
    suma = suma + ++n;
```

lub

```
while ((suma = suma + ++n) < max);
```

lub

```
while ((suma += ++n) < max);
```

Pętla **while** jest często stosowana do wielokrotnego wykonywania tego samego fragmentu programu. Poniższy program sprawdza parzystość kolejnych liczb wprowadzanych przez użytkownika. Zakończenie działania programu nastąpi po wprowadzeniu dowolnego znaku (np. litery) zamiast liczby całkowitej.

Program sprawdzający parzystość kolejnych liczb wprowadzanych z klawiatury.

```
#include <stdio.h>
#pragma warning(disable:4996)

int main(void)
{
    int x;

    printf("Podaj liczbe (znak - koniec): ");
    while (scanf("%d",&x) == 1)
    {
        if (x == 0)
            printf("%d - zero\n",x);
        else
            if (x % 2 == 0)
                printf("%d - liczba parzysta\n",x);
            else
                printf("%d - liczba nieparzysta\n",x);
        printf("Podaj liczbe (znak - koniec): ");
    }

    return 0;
}
```

Przykładowe wywołanie programu:

```
Podaj liczbe (znak - koniec): -3
-3 - liczba nieparzysta
Podaj liczbe (znak - koniec): -2
-2 - liczba parzysta
Podaj liczbe (znak - koniec): 0
0 - zero
Podaj liczbe (znak - koniec): 3
3 - liczba nieparzysta
Podaj liczbe (znak - koniec): x
```

Wczytywanie liczby zostało umieszczone bezpośrednio w pętli **while**. Funkcja **scanf()** zwraca liczbę prawidłowo dokonanych przypisań. Jeśli użytkownik wprowadzi liczbę całkowitą, to **scanf()** zwróci wartość **1** i pętla zostanie wykonana. Wprowadzenie innych znaków spowoduje zwrócenie wartości **0** i zakończenie pętli.

Działanie pętli **while** może zostać przerwane instrukcją **break**. W pętli **while** można zastosować również instrukcję **continue**. Spowoduje ona przerwanie bieżącej iteracji pętli i przejście do sprawdzenia wartości **wyr**.

Po nawiasie w instrukcji **while** nie stawia się średnika. Konstrukcja ze średnikiem na końcu jest poprawna składniowo (kompilator nie zasygnalizuje błędu), ale oznacza wykonywanie w pętli **instrukcji pustej**. Natomiast właściwa **instrukcja** zostanie wykonana tylko raz. Dodatkowo, jeśli **instrukcja** wpływa na wartość **wyr**, to program może „zapęlić się”, tzn. powstanie pętla nieskończona.

```
i = 0;

while (i < 10);
    printf("%d\n", i++);
```

Innym rodzajem błędu jest brak modyfikacji wartości zmiennych występujących w wyrażeniu **wyr**. Spowoduje to także powstanie pętli nieskończonej.

```
i = 0;

while (i < 10)
    printf("%d\n", i);
```

W pewnych sytuacjach celowo tworzy się pętlę nieskończoną. W takim przypadku jej opuszczenie może nastąpić poprzez użycie instrukcji **break**.

Poniższy program oblicza sumę liczb parzystych wprowadzanych z klawiatury. Zakończenie programu następuje po wprowadzeniu pierwszej liczby nieparzystej.

Program obliczający sumę parzystych liczb wprowadzanych z klawiatury.

```
#include <stdio.h>
#pragma warning(disable:4996)
```

```

int main(void)
{
    int x, suma = 0;

    while (1)
    {
        printf("Podaj liczbe: ");
        scanf("%d", &x);
        if (x % 2 == 0)
            suma = suma + x;
        else
            break;
    }
    printf("Suma liczb parzystych: %d\n", suma);

    return 0;
}

```

Opuszczenie pętli można zrealizować poprzez wykorzystanie tzw. zmiennej kontrolnej. W poniższym programie funkcję tę pełni zmienna **koniec**.

Program obliczający sumę parzystych liczb wprowadzanych z klawiatury.

```

#include <stdio.h>
#pragma warning(disable:4996)

int main(void)
{
    int x, suma = 0, koniec = 0;

    while (!koniec)
    {
        printf("Podaj liczbe: ");
        scanf("%d", &x);
        if (x % 2 == 0)
            suma = suma + x;
        else
            koniec = 1;
    }
    printf("Suma liczb parzystych: %d\n", suma);

    return 0;
}

```

Przykładowe wywołanie programu:

```
Podaj liczbe: 4
Podaj liczbe: 8
Podaj liczbe: 2
Podaj liczbe: 1
Suma liczb parzystych: 14
```

Pętlę **while** można wykorzystać do zastąpienia pętli **for**. Szczególne znaczenie ma wtedy kolejność wykonywania wyrażeń **wyr1**, **wyr2** i **wyr3**.

```
for (wyr1; wyr2; wyr3)
    instrukcja;
```

```
wyr1;
while (wyr2)
{
    instrukcja;
    wyr3;
}
```

Przykład zastąpienia pętli **for** przez pętlę **while**:

```
for (i = 0; i < 10; i++)
    printf("%d\n", i);
```

```
i = 0;
while (i < 10)
{
    printf("%d\n", i);
    i++;
}
```

Możliwa jest także sytuacja odwrotna - zastąpienie pętli **while** pętlą **for**.

```
while (wyr)
    instrukcja;
```

```
for ( ; wyr; )
    instrukcja;
```

2.9. Instrukcja **do...while**

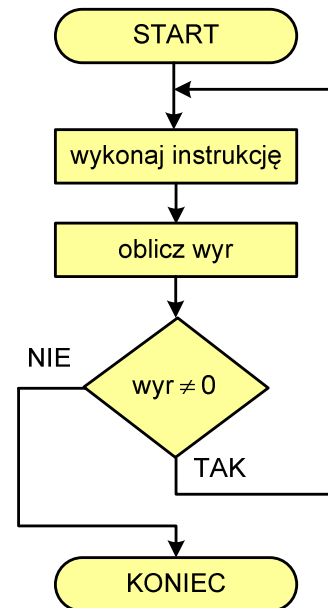
Zasada działania pętli **do...while** jest taka sama jak instrukcji **while**, ale z tą różnicą, że warunek (wartość **wyr**) sprawdzany jest po wykonaniu **instrukcji**.

Składnia tej instrukcji jest następująca:

```
do
    instrukcja;
while (wyr);
```

Zasada działania:

1. Wykonywana jest **instrukcja**.
2. Obliczana jest wartość **wyr**. Jeśli wartość **wyr** jest różna od zera (wyrażenie jest prawdziwe), to następuje powrót do pkt. 1 (wykonanie **instrukcji**). Jeśli wartość **wyr** jest równa zeru (wyrażenie nie jest prawdziwe), to pętla **do...while** kończy działanie.



Działanie pętli **do...while** można opisać także w następujący sposób: „wykonuj instrukcję dopóki wyrażenie w nawiasach jest prawdziwe”. Pozostałe uwagi są takie same jak dla pętli **while** (zastosowanie **break** i **continue**, użycie instrukcji grupującej). W poniższym przykładzie pętla **do...while** jest wykonywana do osiągnięcia przez zmienną **i** wartości **10**.

```
int i = 0;

do
    i++;
while (i < 10);
```

Podstawowa różnica pomiędzy pętlami **while** i **do...while** polega na tym, że w pętli **while** **instrukcja** może nie być ani razu wykonana, natomiast w pętli **do...while** jest ona zawsze wykonywana przynajmniej jeden raz.

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać wybrane zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane różne zadania.

1. Napisz program wyświetlający na ekranie poniższe liczby. Do wyświetlenia jednego wiersza liczb zastosuj jedną pętlę **for**.

```
1  2  3  4  5  6  7  8  9
9  8  7  6  5  4  3  2  1
2  4  6  8  10 12 14 16
-12 -8  -4  0  4  8  12
1  2  4  8  16 32 64 128
11  9.5  8  6.5  5  3.5  2  0.5
```

2. Napisz program obliczający i wyświetlający wartość silni liczby n wprowadzonej z klawiatury.
3. Rezystancję przewodu w zależności od temperatury opisuje wzór (1).

$$R_t = R_{20} [1 + \alpha(t - 20^\circ\text{C})] \quad (1)$$

gdzie:

R_t - rezystancja w temperaturze t ,

R_{20} - rezystancja w temperaturze 20°C ,

α - współczynnik temperaturowy rezystancji w $^\circ\text{C}^{-1}$.

Przewód miedziany ($\alpha = 4,3 \cdot 10^{-3}$) ma w temperaturze $t = 20^\circ\text{C}$ rezystancję $R_{20} = 10 \Omega$. Napisz program obliczający i wyświetlający rezystancję tego przewodu w temperaturze od 0°C do 200°C z krokiem 20°C .

4. Napisz program, w którym użytkownik wprowadza dwie liczby całkowite określające dolną i górną granicę przedziału. Program powinien wyświetlić wszystkie liczby z tego przedziału, ich kwadraty i sześciany. Przykład:

```
Dolna granica:  2
Gorna granica:  5
```

2	4	8
3	9	27
4	16	64
5	25	125

5. Ciąg Fibonacciego opisany jest następującym wzorem rekurencyjnym:

$$F_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ F_{n-1} + F_{n-2} & \text{dla } n > 1 \end{cases} \quad (2)$$

Napisz program obliczający wartość **n**-tego wyrazu tego ciągu.

6. Napisz program działający w pętli, który wczytuje liczbę rzeczywistą typu **float**. Jeśli liczba jest większa od zera, to program oblicza i wyświetla jej pierwiastek kwadratowy, natomiast jeśli liczba jest mniejsza od zera, to program oblicza i wyświetla jej kwadrat. Jeśli liczba jest równa zeru, to program kończy działanie, w przeciwnym przypadku prosi o wprowadzenie kolejnej liczby, itd.

Przykład działania programu:

```

Podaj liczbę: 3
Pierwiatek: 1.732051

Podaj liczbę: -4
Kwadrat: 16.000000

Podaj liczbę: -2
Kwadrat: 4.000000

Podaj liczbę: 9
Pierwiatek: 3.000000

Podaj liczbę: 0
KONIEC

```

7. Napisz program wczytujący liczby całkowite tak długo, aż użytkownik wprowadzi liczbę zero. Następnie program wyświetla ilość wczytanych liczb nieparzystych.

Przykład uruchomienia programu:

```

3
-2
4
7
-5
0
-----
Liczby nieparzyste: 3

```

8. Napisz program wczytujący liczby całkowite tak długo, dopóki tworzą one ciąg rosnący. Następnie program wyświetla sumę wszystkich liczb tworzących ten ciąg (czyli bez ostatniej wartości).

Przykłady uruchomienia programu:

2	-2	2
4	0	1
5	3	-----
8	4	Suma: 2
7	4	
-----	-----	
Suma: 19	Suma: 5	

9. Wyraz ogólny a_n szeregu liczbowego ma postać (3).

$$a_n = \frac{1,25 \cdot (n + 2)}{n^2} \quad (3)$$

Napisz program obliczający sumę S tego szeregu.

$$S = a_1 + a_2 + a_3 + \dots = \sum_{n=1}^{\infty} a_n \quad (4)$$

Zgodnie ze wzorem (4) sumowaniu powinno podlegać nieskończenie wiele wyrazów tego szeregu. Jednakże można zauważyć, że wartości kolejnych wyrazów (przy zwiększającym się n) są coraz mniejsze. W praktyce sumowanie kończy się, gdy wartość kolejnego wyrazu jest mniejsza od założonej dokładności ϵ . Oblicz sumę szeregu S zakładając dokładność $\epsilon = 10^{-2}$.

10. Napisz program obliczający sumę cyfr liczby naturalnej wprowadzonej z klawiatury.

Przykład uruchomienia programu:

```
Podaj liczbe: 4721
-----
Suma cyfr: 14
```

11. Napisz program obliczający liczbę cyfr w liczbie całkowitej wprowadzonej z klawiatury. Zastosuj pętlę **do...while**.

4. Literatura

- [1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.
- [2] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.
- [3] Deitel P.J., Deitel H.: Język C. Solidna wiedza w praktyce. Wydanie VIII. Helion, Gliwice, 2020.
- [4] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.
- [5] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.
- [6] <http://www.cplusplus.com/reference/clibrary> - C library - C++ Reference
- [7] <https://cpp0x.pl/dokumentacja/standard-C/1> - Standard C

5. Pytania kontrolne

1. Scharakteryzuj operatory inkrementacji i dekrementacji.
2. Omów składnię i zastosowanie pętli **for**.
3. Wyjaśnij, jaką rolę w pętli **for** mogą pełnić instrukcje **break**, **goto**, **continue**.
4. Na dowolnym przykładzie opisz sposób wykonywania zagnieżdżonych pętli **for**.

5. Omów składnię i zastosowanie pętli **while**.
6. Omów składnię i zastosowanie pętli **do..while**.
7. Jakie są różnice pomiędzy pętlami **while** i **do..while**?

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.
- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.
- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.
- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.

- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.
- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.