Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Programowanie mikrokontrolerów w języku wysokiego poziomu 1

Kod przedmiotu: **TS1F1008**

(studia stacjonarne)

JĘZYK C - INSTRUKCJE ITERACYJNE WHILE I DO...WHILE

Numer ćwiczenia

PMC_06

Autor:
dr inż. Jarosław Forenc

Białystok 2024

Spis treści

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows 10/11.

1.2. Oprogramowanie

Na komputerach zainstalowany jest edytor kodu źródłowego Visual Studio Code 1.92 (lub nowszy) wraz z odpowiednimi rozszerzeniami (C/C++, Code Runner, Polish Language Pack for Visual Studio Code) oraz MinGW - zestaw kompilatorów różnych języków programowania (m.in. C, C++, Fortran, Java).

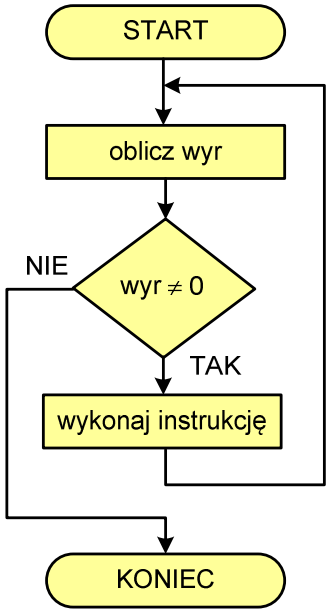
2. Wiadomości teoretyczne

2.1. 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 zeru (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>

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

    return 0;
}
```

Przykładowy wynik uruchomienia programu:

```
Podaj max:     100
Ilosc 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.

```c
#include <stdio.h>

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 „zapętlić 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>
```

```c
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.

```c
#include <stdio.h>

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;
```

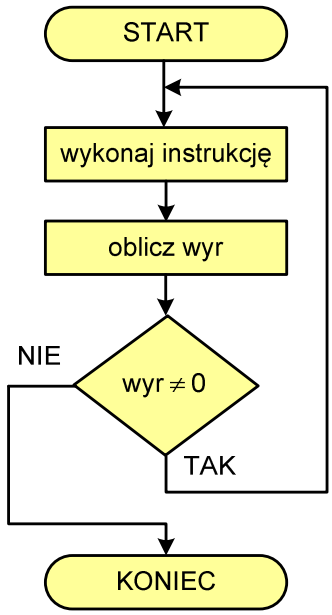
```
for ( ; wyr ; )
    instrukcja;
```

2.2. 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 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 wypadku prosi o wprowadzenie kolejnej liczby, itd.

   Przykład uruchomienia programu:

```
Podaj liczbe: 3
Pierwiatek:   1.732051

Podaj liczbe: -4
Kwadrat:      16.000000

Podaj liczbe: -2
Kwadrat:      4.000000

Podaj liczbe: 9
Pierwiatek:   3.000000

Podaj liczbe: 0
KONIEC
```

2. 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
```

3. 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
```

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

   Przykład uruchomienia programu:

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

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

6. Wyraz ogólny $\mathbf{a_n}$ szeregu liczbowego ma postać (1)**.**

$$a_n = \frac{1{,}25 \cdot (n+2)}{n^2} \quad dla \quad n > 0 \tag{1}$$

   Napisz program obliczający sumę **S** tego szeregu.

$$S = a_1 + a_2 + a_3 + \ldots = \sum_{n=1}^{\infty} a_n \tag{2}$$

   Zgodnie ze wzorem (2) 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 **eps**. Oblicz sumę szeregu **S** zakładając dokładność **eps = 10^{-2}**.

7. Napisz program sprawdzający rodzaj wciśniętego znaku na klawiaturze. Program powinien działać w pętli i rozpoznawać litery, cyfry, znaki odstępu oraz inne znaki. Wyjście z programu powinno nastąpić po wciśnięciu litery '**q**' lub '**Q**'. Przykłady działania programu:

```
Wcisnales litere: k
Wcisnales cyfre: 7
Wcisnales inny znak: *
```

Wykorzystaj funkcje znajdujące się w pliku nagłówkowym **ctype.h**:

- **isdigit(znak)** - zwraca wartość różną od zera, gdy **znak** jest cyfrą;
- **isalpha(znak)** - zwraca wartość różną od zera, gdy **znak** jest literą;
- **isspace(znak)** - zwraca wartość różną od zera, gdy **znak** jest odstępem.

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

[8] https://code.visualstudio.com/ - Visual Studio Code

[9] https://sourceforge.net/projects/mingw/ - MinGW

5. Pytania kontrolne

1. Omów składnię i zastosowanie pętli **while**.
2. Omów składnię i zastosowanie pętli **do..while**.
3. Wyjaśnij, 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.