Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Programowanie w języku C

Kod przedmiotu: **TZ1E1004BK**

(studia niestacjonarne)

JĘZYK C - OPERACJE WE-WY, ZMIENNE, TYPY I NAZWY ZMIENNYCH, OPERATORY I WYRAŻENIA ARYTMETYCZNE, FUNKCJE MATEMATYCZNE

Numer ćwiczenia

PROGwC_02

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 zainstalowane jest środowisko programistyczne Code::Blocks.

2. Wiadomości teoretyczne

2.1. Zastosowanie zmiennych w programie

Głównym zadaniem przedstawianych do tej pory programów było wyświetlanie tekstu. Teraz zostanie napisany prosty program wykonujący operacje arytmetyczne i wykorzystujący zmienne przechowujące wartości.

Zadaniem programu będzie zamiana temperatury podanej w skali Fahrenheita na temperaturę w skali Celsjusza. Kolejność wykonywania operacji w programie jest następująca:

- użytkownik podaje temperaturę w skali Fahrenheita;
- program oblicza temperaturę w skali Celsjusza według wzoru:

$$Tc = \frac{5}{9}(Tf - 32) \tag{1}$$

- program wyświetla obliczoną temperaturę w skali Celsjusza.

W programie będą występowały dwie wartości (temperatura w skali Fahrenheita i temperatura w skali Celsjusza), a zatem należy wprowadzić dwie zmienne. Kod programu zamieszczono poniżej.

Zamiana temperatury podanej w skali Fahrenheita na temperaturę w skali Celsjusza.

```
#include <stdio.h>                                          1

int main(void)
{
    float tempf; /* temperatura w skali Fahrenheita */      2
    float tempc; /* temperatura w skali Celsjusza */

    printf("Temperatura [F]: ");                            3
    scanf("%f",&tempf);                                     4
    tempc = 5 * (tempf - 32) / 9;                           5
    printf("Temperatura [C]: %f\n",tempc);                  6

    return 0;                                               7
}
```

Przykładowy wynik uruchomienia programu:

```
Temperatura [F]: 75
Temperatura [C]: 23.888889
```

Opis kodu programu:

1 - dołączenie pliku nagłówkowego:

stdio.h - zawiera deklaracje funkcji **printf()** i **scanf()**;

2 - deklaracja dwóch zmiennych: **tempf** i **tempc** będących liczbami rzeczywistymi (typ **float**);

3 - wyświetlenie napisu: **Temperatura [F]:** - bez znaku **\n** na końcu;

4 - wczytanie temperatury w skali Fahrenheita:

tempf - nazwa zmiennej;

&tempf - adres zmiennej (**scanf()** wymaga podania adresu zmiennej);

%f - określa typ wczytywanej zmiennej (**%f** - typ **float**);

5 - obliczenie wartości wyrażenia arytmetycznego;

6 - wyświetlenie wyniku czyli łańcucha znaków: **Temperatura [C]:** i wartości zmiennej **tempc**; w miejscu, w którym ma być wyświetlona wartość zmiennej podajemy specyfikator formatu - **%f**, podczas wyświetlania będzie on

zastąpiony wartością zmiennej, której nazwę podajemy po cudzysłowie kończącym łańcuch znaków i po przecinku;

7 - zakończenie programu.

2.2. Zmienne

Zmienne służą do reprezentacji (przechowywania) wartości danych, które mogą być zmieniane podczas działania programu. Zbiór wartości, jakie mogą przyjmować zmienne nazywa się **typem** (np. liczby całkowite, rzeczywiste). Zmienne przechowywane są w pamięci komputera. Każda zmienna (poza nazwą) ma adres (komputer nie posługuje się nazwami zmiennych tylko ich adresami).

Przed wykorzystaniem zmiennej w programie należy wcześniej ją zadeklarować, czyli podać jej **typ** i **nazwę**:

```
typ nazwa;
```

Na końcu deklaracji stawia się średnik. W poniższym przykładzie **int** jest nazwą typu, zaś **a** - nazwą zmiennej.

```
int a;
```

Gdy jest kilka zmiennych tego samego typu, to można je deklarować po przecinku.

```
int a;
float b, c;
```

Umieszczenie deklaracji każdej zmiennej w oddzielnej linii jest wygodne, gdy dodajemy **komentarze** opisujące przeznaczenie poszczególnych zmiennych.

```
float d;  /* zmienna d */
float e;  /* zmienna e */
```

2.3. Typy zmiennych

Podstawowe typy zmiennych w języku C zostały zestawione w Tabeli 1.

Tabela 1. Podstawowe typy zmiennych w języku C

Nazwa typu	Zakres wartości danych	Rozmiar (bajty)	Uwagi
`char`	-128 ... 127	1	małe liczby całkowite, znaki ASCII
`int`	-2147483648 ... 2147483647	4	liczby całkowite
`float`	$-3{,}4\cdot10^{38}$ … $3{,}4\cdot10^{38}$	4	liczby rzeczywiste, 7 cyfr znaczących
`double`	$-1{,}7\cdot10^{308}$ … $1{,}7\cdot10^{308}$	8	liczby rzeczywiste, 15 cyfr znaczących
`void`	-	-	oznacza brak wartości

Dodatkowo istnieją cztery słowa kluczowe modyfikujące powyższe typy.

- dla liczb całkowitych:

  signed, **unsigned** - określa czy zmienna ma być ze znakiem czy bez;

  short, **long** - dla typu **int** oznacza krótką lub długa liczbę całkowitą.

- dla liczb rzeczywistych:

  long - dla typu **double** zwiększa precyzję (liczbę miejsc po przecinku).

Stosując powyższe słowa kluczowe otrzymujemy dodatkowe typy (Tabela 2).

Tabela 2. Typy zmiennych w języku C

Nazwa typu	Zakres wartości danych	Rozmiar (bajty)	Uwagi
`signed char = char`	-128 ... 127	1	liczby całkowite
`unsigned char`	0 … 255	1	liczby całkowite
`short = signed short int`	-32 768 … 32 767	2	liczby całkowite
`unsigned short = unsigned short int`	0 … 65 535	2	liczby całkowite
`signed int = int`	-2 147 483 648 … 2 147 483 647	4	liczby całkowite

unsigned = unsigned int	0 … 4 294 967 295	4	liczby całkowite
long = signed long int	-2 147 483 648 … 2 147 483 647	4	liczby całkowite
unsigned long = unsigned long int	0 … 4 294 967 295	4	liczby całkowite
long long = signed long long int	-9 223 372 036 854 775 808 … 9 223 372 036 854 775 807	8	liczby całkowite
unsigned long long = unsigned long long int	0 … 18 446 744 073 709 551 615	8	liczby całkowite
float	$-3{,}4\cdot10^{38}$ … $3{,}4\cdot10^{38}$	4	7 cyfr znaczących
double	$-1{,}7\cdot10^{308}$ … $1{,}7\cdot10^{308}$	8	15 cyfr znaczących
long double	$-1{,}7\cdot10^{308}$ … $1{,}7\cdot10^{308}$	8	15 cyfr znaczących
void	-	-	-

Powyższe zakresy i rozmiary podane są dla środowiska Microsoft Visual Studio 2008. W zależności od kompilatora mogą wystąpić różnice w rozmiarze zmiennych typu **int** i **long double** (Tabela 3). Zakresy dla poszczególnych typów zapisane są w pliku nagłówkowym **limits.h**.

Tabela 3. Liczba bajtów zajmowanych przez zmienne typów **int** i **long double** zależnie od środowiska programistycznego

Kompilator	int (bajty)	long double (bajty)
Borland C++ 3.1	2	10
Dev-C++	4	12
Microsoft Visual Studio 2008	4	8
Code::Blocks 20.03	4	16
Borland C++ Builder 6	4	10

2.4. Nazwy zmiennych

Nazwa zmiennej może składać się z liter, cyfr i znaku podkreślenia:

```
a  b  c  d  e  f  g  h  i  j  k  l  m  _
n  o  p  q  r  s  t  u  v  w  x  y  z
A  B  C  D  E  F  G  H  I  J  K  L  M
N  O  P  Q  R  S  T  U  V  W  X  Y  Z
0  1  2  3  4  5  6  7  8  9
```

Pierwszym znakiem nazwy musi być litera. Znak podkreślenia traktowany jest jak litera. Nie zaleca się rozpoczynania nazwy zmiennej od znaku podkreślenia, gdyż takie nazwy często występują w programach bibliotecznych. W nazwach zmiennych nie stosuje się znaków spacji. Przyjęło się, że nazwy zmiennych pisze się małymi literami, a nazwy stałych - wielkimi.

Nazwa zmiennej powinna być związana z jej zawartością. Długość nazwy nie jest ograniczona, ale rozróżnialne są 63 pierwsze znaki. Jako nazw zmiennych nie można stosować słów kluczowych języka C:

```
auto        extern      short       while
break       float       signed      _Alignas
case        for         sizeof      _Alignof
char        goto        static      _Bool
const       if          struct      _Complex
continue    inline      switch      _Generic
default     int         typedef     _Imaginary
do          long        union       _Noreturn
double      register    unsigned    _Static_assert
else        restrict    void        _Thread_local
enum        return      volatile
```

2.5. Stałe liczbowe

Stałe liczbowe są to liczby zapisane bezpośrednio w kodzie programu. Typ liczby zależny jest od formy zapisu i wartości liczby.

W przypadku liczb całkowitych domyślnym typem jest **int**. Jeśli wartość liczby przekracza zakres tego typu, to dana liczba jest traktowana jako **long int**, **unsigned long int**, **long long int** lub **unsigned long long int**, np.:

`1` - stała całkowita typu **int**;

`25000` - stała całkowita typu **int**;

`39000` - stała całkowita typu **int** (4 bajty) lub **long**;

`4100000000` - stała typu **unsigned long int** (bo przekracza typ **long int**).

Na końcu liczby całkowitej mogą pojawić się dodatkowe litery zmieniające jej typ:

- **u** lub **U** zmienia typ na **unsigned** (**unsigned int** lub **unsigned long**);
- **l** lub **L** zmienia typ na **long** (**long int** lub **unsigned long int**);
- **ll** lub **LL** zmienia typ na **long long** (**long long int** lub **unsigned long long int**).

Przykład:

`5U` - stała całkowita typu **unsigned int**;

`5L` - stała całkowita typu **long**;

`10ul` - stała całkowita typu **unsigned long**;

`6llu` - stała całkowita typu **unsigned long long**.

Domyślnie liczby całkowite zapisywane są w systemie dziesiętnym. Liczby w systemie **ósemkowym** zaczynają się od **0** (zera), zaś liczby w systemie **szesnastkowym** zaczynają się od **0x** lub **0X**, np.:

`011` - **11** w systemie **ósemkowym** to **9** w systemie **dziesiętnym**;

`0x11` - **11** w systemie **szesnastkowym** to **17** w systemie **dziesiętnym**.

Liczby w systemie ósemkowym i szesnastkowym są traktowane jako wartości typu **unsigned int** (ewentualnie **unsigned long** lub **unsigned long long**).

Domyślnym typem dla liczb rzeczywistych (zmiennoprzecinkowych) jest **double**, np.:

`1.0` - stała rzeczywista typu **double** (wartość: 1);

`1.312e+2` - stała rzeczywista typu **double** (wartość: $1{,}312 \cdot 10^{2}$);

`-2.124E-1` - stała rzeczywista typu **double** (wartość: $-2{,}124 \cdot 10^{-1}$).

W zapisie liczby rzeczywistej:

- zawsze można pominąć znak plus, np. `1.312e2`

- można pominąć kropkę dziesiętną, np. `2E5`
- można pominąć część wykładniczą, np. `15.21`
- można pominąć część ułamkową, np. `3.e14`
- można pominąć część całkowitą, np. `.21e-5`
- nie wolno używać odstępów (spacji), np. `1.23 E + 5`

Na końcu liczby rzeczywistej mogą pojawić się dodatkowe litery zmieniające jej typ:

- **l** lub **L** zmienia typ na **long double**;
- **f** lub **F** zmienia typ na **float**.

Przykład:

`2.5L` - stała rzeczywista typu **long double**;
`4.52f` - stała rzeczywista typu **float.**

2.6. Operatory i wyrażenia arytmetyczne, operator przypisania

W języku C występują dwa podstawowe operatory jednoargumentowe:

+ - plus, jako znak liczby (zazwyczaj jest pomijany);
– - minus, jako znak liczby.

Do operatorów dwuargumentowych zalicza się:

+ - dodawanie;
– - odejmowanie;
***** - mnożenie;
/ - dzielenie (dla liczb całkowitych obcina część ułamkową);
% - dzielenie modulo (reszta z dzielenia, tylko dla typów całkowitych).

Operator przypisania **=** (znak równości) stosowany jest do nadania wartości zmiennej. Poniższe wyrażenie powinno być interpretowane jako: weź wartość numeryczną **10** i umieść ją w pamięci w miejscu skojarzonym ze zmienną **a**. Operatora przypisania nie należy kojarzyć ze znakiem równości.

```
a = 10;
```

Zapis:

```
a = a + 10;
```

matematycznie nie jest poprawny. W programie w języku C należy interpretować go jako: pobierz wartość znajdującą się w pamięci w miejscu skojarzonym ze zmienną **a**, dodaj do tej wartości liczbę **10** i otrzymany wynik umieść z powrotem w pamięci w miejscu skojarzonym ze zmienną **a**.

W języku C prawidłowy jest także poniższy zapis:

```
a = b = c = d + 10;
```

oznacza on: weź wartość zmiennej **d** dodaj do niej **10**, otrzymaną wartość przypisz zmiennej **c**, następnie zmiennej **b** przypisz wartość zmiennej **c**, a zmiennej **a** przypisz wartość zmiennej **b**. Powyższy zapis jest zatem równoważny instrukcji:

```
a = (b = (c = (d + 10)));
```

Operatory arytmetyczne są lewostronnie łączne. Oznacza to, że jeśli obok siebie występują dwa operatory o takim samym priorytecie, to jako pierwsze wykonywane jest działanie znajdujące się po lewej stronie. W poniższym przykładzie jako pierwsze zostanie wykonane mnożenie **a * b**.

```
z = a * b * c;
```

Powyższy zapis jest zatem równoważny instrukcji:

```
z = ((a * b) * c);
```

Spośród poznanych dotąd operatorów najwyższy priorytet mają jednoargumentowe operatory **+** i **–** (znaki liczb), następnie są operatory

***** (mnożenie), **/** (dzielenie), **%** (dzielenie modulo). Niższy priorytet ma dodawanie (**+**) i odejmowanie (**-**), natomiast najniższy - operator przypisania (**=**).

Zastosowanie nawiasów zmienia priorytet operatorów. Jeśli nie jesteśmy pewni kolejności wykonywania działań zawsze używajmy dodatkowych nawiasów zwykłych **(** i **)**. Mogą to być wielokrotne nawiasy zwykłe. W wyrażeniach arytmetycznych nie wolno natomiast jako nawiasy stosować symboli: **[ ] { }**.

Wyrażenia arytmetyczne mogą zawierać operatory arytmetyczne jednoargumentowe, dwuargumentowe, nawiasy zwykłe oraz wywołania funkcji. Każde wyrażenie arytmetyczne ma **wartość** i **typ**.

Wartość początkową zmiennej można nadać już podczas jej deklaracji - operacja taka nazywa się inicjalizacją, np.

```
int main(void)
{
    int a = 0;          /* deklaracja z inicjalizacją */
    int b;              /* deklaracja bez inicjalizacji */
    float c = -5.5f;    /* deklaracja z inicjalizacją */
    float d;            /* deklaracja bez inicjalizacji */

    b = 15;             /* przypisanie wartości */
    d = 1.6e-4f;        /* przypisanie wartości */

    return 0;
}
```

2.7. Złożone (skrócone) operatory przypisania

Wyrażenia modyfikujące (aktualizujące) wartość pewnej zmiennej mogą być zapisywane w skrócony sposób poprzez użycie tzw. złożonych operatorów przypisania:

```
x = x + 1;
z = z - 2;
y = y * 10.5;
s = s / 3;
```

jest równoważne:

```
x += 1;
z -= 2;
y *= 10.5;
s /= 3;
```

Jeśli **w1** i **w2** są wyrażeniami, to:

```
w1 = (w1) operator (w2);
```

jest równoważne:

```
w1 operator= w2;
```

Ten sposób zapisu można stosować dla 10 operatorów:

```
+   -   *   /   %   <<   >>   &   ^   |
```

2.8. Dyrektywa preprocesora #define

Dyrektywa preprocesora **#define** służy do definiowania stałych (tzw. stałych symbolicznych). Umieszczana jest zazwyczaj bezpośrednio po dyrektywach **#include**. Wyrażenia stałe pisze się najczęściej wielkimi literami.

Program zamieniający podaną kwotę w złotych (PLN) na dolary (USD) i euro (EUR).

```
#include <stdio.h>
#define USD 4.1936f
#define EUR 4.3869f

int main(void)
{
    float pln, usd, eur;

    printf("Podaj kwote w PLN: ");
    scanf("%f",&pln);
    usd = pln / USD;
    eur = pln / EUR;
    printf("%.2f PLN to %.2f USD\n",pln,usd);
    printf("%.2f PLN to %.2f EUR\n",pln,eur);

    return 0;
}
```

Przykładowy wynik uruchomienia programu:

```
Podaj kwote w PLN: 100
100.00 PLN to 23.85 USD
100.00 PLN to 22.80 EUR
```

Wyrażenie stałe znajdujące się po dyrektywie **#define** składa się z dwóch części: nazwy wyrażenia (**USD**) oraz jej wartości (**3.7273**). Wyrażenia stałe są obliczane na etapie prekompilacji programu, a nie podczas jego wykonania. W trakcie prekompilacji każde wystąpienie stałej **USD** jest zastępowane jej wartością (czyli liczbą **3.7273)**.

2.9. Funkcja printf()

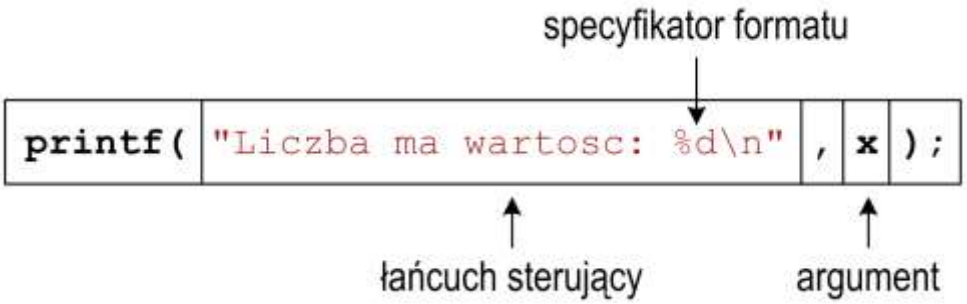
Funkcja **printf()** ma następującą składnię:

```
printf("łańcuch sterujący",argument1,argument2,…);
```

Funkcja **printf()** wyświetla tekst na ekranie. Gdy w łańcuchu sterującym występuje specyfikator formatu zaczynający się od znaku procentu (**%**), wówczas następuje przekształcenie, tj. w miejsce specyfikatora wstawiana jest wartość argumentu. Jako argument może występować zmienna, stała liczbowa, wyrażenie lub wywołanie funkcji zwracającej wartość. W poniższym przykładzie wyświetlana jest wartość zmiennej **x** typu **int**.

```
int x = 15;
printf("Liczba ma wartosc: %d\n",x);
```

Na Rys. 1 przedstawione są elementy składowe funkcji **printf()**.



Rys. 1. Struktura funkcji **printf()**

Specyfikator formatu określa **typ** oraz **sposób wyświetlania** argumentu na ekranie. Liczba specyfikatorów formatu musi być zgodna z liczbą argumentów.

Jeśli typ argumentu zostanie błędnie określony to na ekranie wyświetlona zostanie nieprawidłowa wartość.

W specyfikatorze formatu zawsze musi występować **znak procentu** (**%**) oraz **typ**. Pozostałe elementy specyfikatora formatu są opcjonalne - mogą wystąpić, ale nie muszą. Nawiasy kwadratowe w poniższym zapisie oznaczają elementy opcjonalne:

specyfikator = %[znacznik][szerokość][.precyzja][modyfikator]typ

[znacznik] - "**+**" - przed liczbą stawiany jest znak (plus lub minus);
"**-**" - wyrównanie wyświetlanych znaków do lewej strony;
" " - (spacja), przed liczbą dodatnią dodaje spację;
"**0**" - wypełnia początkowe pola zerami zamiast spacjami;
"**#**" - poprzedza liczby w systemie ósemkowym zerem (**0**), zaś w systemie szesnastkowym - **0x**;

[szerokość] - określa minimalną liczbę wyprowadzanych znaków, jeśli znaków jest mniej to pole jest z lewej strony uzupełniane spacjami, jeśli więcej - podana szerokość jest ignorowana; w przypadku łańcucha znaków (**%s**) określa maksymalną liczbę wyświetlanych znaków;

[.precyzja] - liczba wyświetlanych cyfr po kropce dziesiętnej;

typ - określa rodzaj i typ argumentu:

- **d, i** - liczba całkowita ze znakiem (**signed**), dziesiętna;
- **u** - liczba całkowita bez znaku (**unsigned**), dziesiętna;
- **x, X** - liczba całkowita bez znaku, szesnastkowa;
- **o** - liczba całkowita bez znaku, ósemkowa;
- **f** - liczba rzeczywista w postaci **[-]ddd.ddd**;
- **e, E** - liczba rzeczywista w formacie „naukowym" (symbol **e** lub **E**);
- **g, G** - liczba rzeczywista (format **f** lub **e**);

s - ciąg znaków;

c - pojedynczy znak;

p - wskaźnik.

[modyfikator] - służy do zmodyfikowania podstawowego typu podawanego przez znak typu:

h - w połączeniu ze specyfikatorem całkowitym oznacza **short int** (**%hd**) lub **unsigned short int** (**%hu**);

hh - w połączeniu ze specyfikatorem całkowitym oznacza **signed char** (**%hhd**) lub **unsigned char** (**%hhu**);

l - w połączeniu ze specyfikatorem całkowitym oznacza **long int** (**%ld**) lub **unsigned long int** (**%lu**);

ll - w połączeniu ze specyfikatorem całkowitym oznacza **long long int** (**%lld**) lub **unsigned long long int** (**%llu**);

L - stosowany do wyświetlania wartości rzeczywistych typu **long double**.

Załóżmy, że mamy w programie następujące deklaracje i inicjalizacje zmiennych:

```
int    i = 15
int    j = -30;
float  x = 15.1234567f;
double y = 1.456e-2;
char   text[10] = "Napis";
```

- wyświetlenie dwóch zmiennych całkowitych (**%d**, **%d**) oraz zmiennych rzeczywistych w formacie „zwykłym” (**%f**) i w formacie naukowym (**%e**):

```
printf("%d %d %f %e",i,j,x,y);
```

```
15 -30 15.123457 1.456000e-002
```

- sposób zapisu liczb rzeczywistych przy inicjalizacji (format „zwykły” lub format naukowy) nie ma wpływu na sposób ich przechowywania w pamięci komputera:

```
printf("%f %e\n",x,x);
printf("%f %e",y,y);
```

```
15.123457 1.512346e+001
0.014560 1.456000e-002
```

- liczba po znaku procentu określa szerokość, czyli ilość pozycji, na których jest wyświetlana liczba; brakujące pozycje są uzupełniane spacjami; znacznik „**+**” powoduje wyświetlenie znaku liczby, a znacznik „**-**” - wyrównanie wyświetlanej liczby do lewej (dodatkowe spacje są wyświetlane za liczbą, a nie przed nią):

```
printf("%5d %+5d %-5d",i,i,i);
```

```
   15  +15 15
```

- w specyfikatorze formatu liczba przed kropką oznacza szerokość, zaś liczba po kropce oznacza precyzję, czyli liczbę znaków po kropce dziesiętnej; szerokość dotyczy **całej liczby** (część całkowita + kropka + część ułamkowa), a nie tylko części całkowitej:

```
printf("%10.3f",x);
```

```
    15.123
```

- jeśli szerokość jest zbyt mała do wyświetlenia liczby, to zostanie przez kompilator zignorowana:

```
printf("%1.5f",x);
```

```
15.12346
```

- specyfikator formatu bez znaku procentu na początku traktowany jest jak zwykły tekst:

```
printf("x = %1.3f, y = 1.3f",x,y);
```

```
x = 15.123, y = 1.3f
```

- do wyświetlenia tekstu używamy specyfikatora formatu **%s**:

```
printf("Tekst: %s",text);
```

```
Tekst: Napis
```

2.10. Funkcja scanf()

Funkcja **scanf()** ma następującą składnię:

```
scanf("specyfikator",argumenty);
```

Funkcja **scanf()** wczytuje znaki ze standardowego wejścia (klawiatura), interpretuje je zgodnie z zadanym specyfikatorem formatu i w odpowiedniej kolejności przypisuje wyniki argumentom.

W specyfikatorze formatu zawsze musi występować znak procentu (**%**) oraz **typ**. Pozostałe elementy specyfikatora formatu są opcjonalne:

specyfikator = %[szerokość][modyfikator]typ

[szerokość] - określa ile znaków zostanie przeczytanych;

typ - określa rodzaj i typ argumentu:

- **d** - liczba całkowita dziesiętna, typ **int**;
- **D** - liczba całkowita dziesiętna, typ **long**;

o - liczba całkowita ósemkowa, typ **int**;

O - liczba całkowita ósemkowa, typ **long**;

x - liczba całkowita szesnastkowa, typ **int**;

X - liczba całkowita szesnastkowa, typ **long**;

i - liczba całkowita dziesiętna, ósemkowa, szesnastkowa, typ **int**;

l - liczba całkowita dziesiętna, ósemkowa, szesnastkowa, typ **long**;

u - liczba całkowita dziesiętna bez znaku, typ **unsigned int**;

U - liczba całkowita dziesiętna bez znaku, typ **unsigned long**;

f, e, E - liczba rzeczywista, typ **float**;

g, G - liczba rzeczywista, typ **float**;

s - ciąg znaków;

c - pojedynczy znak, typ **char**;

p - wskaźnik;

[modyfikator] - służy do zmodyfikowania podstawowego typu podawanego przez znak typu:

l - zmienia wszystkie typy całkowitoliczbowe na ich długie wersje; zastosowany do znaków typu **f**, **e**, **E**, **g**, **G** spowoduje interpretację zawartości pól wejściowych jako liczb typ **double**;

L - zastosowany do znaków typu **f**, **e**, **E**, **g**, **G** spowoduje interpretację zawartości pól wejściowych jako liczb typ **long double**;

h - typy całkowitoliczbowe będą traktowane jako **short**.

Argumenty funkcji **scanf()** są adresami obszarów w pamięci, dlatego też muszą być poprzedzone znakiem **&** (nie dotyczy ciągu znaków).

Załóżmy, że mamy w programie następujące deklaracje zmiennych:

```
int    a, b, c;
float  x, z;
double y;
char   text[15];
```

- w przypadku funkcji **scanf()** wczytywane argumenty mogą być oddzielone od siebie dowolną liczbą tzw. białych znaków (**spacja**, **tabulacja**, **enter**). Wczytanie trzech liczb typu **int** może zatem odbyć się w różny sposób:

```
scanf("%d %d %d", &a, &b, &c);
```

```
15□20□-30<enter>
lub
□□□15□□□□20□□-30□□<enter>
lub
15<enter>
20<enter>
-30<enter>
```

- wczytanie liczb typu **int**, **float** i **double**:

```
scanf("%d %f %lf", &a, &x, &y);
```

```
15□1.51□-12.467<enter>
```

- wczytanie dwóch liczb typu **float** (format „zwykły” i naukowy) oraz liczby typu **double** (format naukowy):

```
scanf("%f %e %le", &x, &z, &y);
```

```
12.1□1.45e-2□-1.34e5<enter>
```

- wczytanie tekstu (zmienna **text** jest tablicą, nazwa tablicy jest adresem jej zerowego elementu, zatem nie jest potrzebny znak **&** przed zmienną **text**):

```
scanf("%s", text);
```

```
napis<enter>
```

- funkcja **scanf()** wczytuje jeden argument do pojawienia się pierwszego białego znaku. W przypadku poniższego tekstu zapamiętane zostanie tylko jedno słowo „**Ala**”:

```
scanf("%s", text);
```

```
Ala ma laptopa<enter>
```

2.11. Funkcje matematyczne z pliku nagłówkowego math.h

W pliku **math.h** znajdują się definicje stałych oraz funkcji matematycznych.

Tabela 4. Definicje stałych w pliku **math.h**.

Nazwa stałej	Wartość	Znaczenie
M_E	2.7182818284590452353 6	e - liczba Eulera
M_LOG2E	1.4426950408889634073 6	$\log_2 e$
M_LOG10E	0.4342944819032518276 51	$\log e$
M_LN2	0.6931471805599453094 17	$\ln 2$
M_LN10	2.3025850929940456840 2	$\ln 10$
M_PI	3.1415926535897932384 6	π - liczba pi
M_PI_2	1.5707963267948966192 3	$\pi/2$
M_1_PI	0.3183098861837906715 38	$1/\pi$
M_SQRT2	1.4142135623730950488 0	$\sqrt{2}$
M_SQRT1_2	0.7071067811865475244 01	$\sqrt{2}/2$

W poniższym programie obliczane jest pole koła o promieniu **r** wprowadzonym z klawiatury. Do obliczenia pola wykorzystywana jest stała **M_PI**.

Program obliczający pole koła o promieniu r.

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

int main(void)
{
    float r, pole;

    printf("Podaj promien kola: "); scanf("%f",&r);
    pole = (float)M_PI * r * r;
    printf("Pole kola: %f\n",pole);

    return 0;
}
```

Przykład uruchomienia program:

```
Podaj promien kola: 5
Pole kola: 78.539818
```

Najważniejsze funkcje matematyczne zdefiniowane w pliku nagłówkowym **math.h**:

abs()	Nagłówek: **int abs(int x);**

- $|x|$ - zwraca wartość bezwzględną argumentu x będącego liczbą całkowitą;

acos()	Nagłówek: **double acos(double x);**

- $\arccos x$ - zwraca arcus cosinus argumentu x;
- argument może przyjmować wartości z przedziału $\langle -1,1 \rangle$;
- funkcja zwraca kąt w radianach z zakresu od 0 do π radianów;

asin()	Nagłówek: **double asin(double x);**

- $\arcsin x$ - zwraca arcus sinus argumentu x;

- argument może przyjmować wartości z przedziału $\langle -1,1 \rangle$;
- funkcja zwraca kąt w radianach z zakresu od $-\pi/2$ do $\pi/2$ radianów;

`atan()`	Nagłówek: **double atan(double x);**

- $\operatorname{arctg} x$ - zwraca arcus tangens argumentu x;
- funkcja zwraca kąt w radianach z zakresu od $-\pi/2$ do $\pi/2$ (radianów);

`atan2()`	Nagłówek: **double atan2(double x, double y);**

- $\operatorname{arctg} x/y$ - zwraca arcus tangens ilorazu argumentów x/y;
- argumenty muszą być różne od zera;
- funkcja zwraca kąt w radianach z zakresu od $-\pi$ do π (radianów);

`ceil()`	Nagłówek: **double ceil(double x);**

- zaokrąglenie argumentu x w górę;
- zwraca najmniejszą liczbę całkowitą większą lub równą argumentowi x;

`cos()`	Nagłówek: **double cos(double x);**

- $\cos x$ - zwraca cosinus argumentu x podanego w radianach;
- funkcja zwraca wartość z przedziału $\langle -1,1 \rangle$;

`cosh()`	Nagłówek: **double cosh(double x);**

- $\cosh x$ - zwraca cosinus hiperboliczny argumentu x podanego w radianach;

`exp()`	Nagłówek: **double exp(double x);**

- $\mathbf{e}^x$ - zwraca liczbę **e** (podstawa logarytmu naturalnego) podniesioną do potęgi argumentu x;

`fabs()`	Nagłówek: **double fabs(double x);**

- $|x|$ - zwraca wartość bezwzględną argumentu x będącego liczbą rzeczywistą;

`floor()`	Nagłówek: **double floor(double x);**

- zaokrąglenie argumentu x w dół;
- zwraca największą liczbę całkowitą mniejszą lub równą argumentowi x;

`log()`	Nagłówek: **double log(double x);**

- $\ln x$ - zwraca logarytm naturalny argumentu x;

`log10()`	Nagłówek: **double log10(double x);**

- $\log x$ - zwraca logarytm dziesiętny argumentu x;

`pow()`	Nagłówek: **double pow(double x, double y);**

- x^y - zwraca x podniesione do potęgi y;

`sin()`	Nagłówek: **double sin(double x);**

- $\sin x$ - zwraca sinus argumentu x podanego w radianach;
- funkcja zwraca wartość z przedziału $\langle -1,1 \rangle$;

`sinh()`	Nagłówek: **double sinh(double x);**

- $\sinh x$ - zwraca sinus hiperboliczny argumentu x podanego w radianach;

`sqrt()`	Nagłówek: **double sqrt(double x);**

- $\sqrt{x}$ - zwraca pierwiastek kwadratowy nieujemnego argumentu x;

tan()	Nagłówek: **double tan(double x);**

- $\mathrm{tg}\, x$ - zwraca tangens argumentu x podanego w radianach;

tanh()	Nagłówek: **double tanh(double x);**

- $\mathrm{tgh}\, x$ - zwraca tangens hiperboliczny argumentu x podanego w radianach.

3. Przebieg ćwiczenia

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

1. Do zacisków rezystora **R = 100 Ω** przyłożono napięcie stałe **U = 8 V**. Oblicz i wyświetl wartość prądu **I** płynącego przez rezystor.

   Przykładowe wywołanie programu:

   ```
   Prad I [A]:  0.08
   ```

2. Przez opornik o rezystancji **R** płynie prąd stały **I**. Napisz program, który obliczy napięcie na oporniku **U** oraz wydzielającą się w nim moc **P**. Wartości rezystancji i prądu wczytaj funkcją **scanf()**.

   Przykładowe wywołanie programu:

   ```
   Podaj R [Om]:     470
   Podaj I [A]:      0.25
   ------------------------
   Napiecie U [V]:  117.5
   Moc P [W]:       29.375
   ```

3. Napisz program obliczający współczynniki **a**, **b** równania prostej:

   $$y = ax + b \tag{2}$$

   przechodzącej przez dwa punkty: $\mathbf{P_1(x_1,y_1)}$, $\mathbf{P_2(x_2,y_2)}$. Współrzędne punktów wczytaj funkcją **scanf()**.

Przykładowe wywołanie programu:

```
Wspolrzedne punktu P1
x1:  0
y1:  2

Wspolrzedne punktu P2
x2:  3
y2:  1
--------------------------
Wspolczynnik a:  -0.333333
Wspolczynnik b:  2.000000
```

4. Zadeklaruj trzy zmienne (**x**, **y**, **z**) typu **int**. Wczytaj wartości tych zmiennych funkcją **scanf()** i oblicz:

$$x+y, \qquad x-y, \qquad x\cdot y, \qquad \frac{x}{y} \qquad \frac{x}{y+z}, \qquad x\cdot\frac{y}{z}, \qquad \sqrt{x}$$

Zwróć szczególnie uwagę na poprawność wykonania operacji dzielenia i pierwiastkowania.

5. Napisz program obliczający częstotliwość rezonansową **fr** układu o rezystancji **R**, indukcyjności **L** i pojemności **C** wprowadzonych z klawiatury.

	Przykładowe uruchomienie programu	Wzór

a)	`Rezystancja R [Om]:  10`
	`Indukcyjnosc L [H]:  0.1`
	`Pojemnosc C [F]:  1.0e-6`
	`---`
	`Czestotliwosc fr [Hz]:  503.54397`
		$f_r = \frac{1}{2\pi\sqrt{LC-(RC)^2}}$ (3)

b)	`Rezystancja R [Om]:  5000`
	`Indukcyjnosc L [H]:  0.02`
	`Pojemnosc C [F]:  4.0e-5`
	`---`
	`Czestotliwosc fr [Hz]:  177.942413`
		$f_r = \frac{1}{2\pi\sqrt{LC-\left(\frac{L}{R}\right)^2}}$ (4)

c)	`Rezystancja R [Om]:  500`
	`Indukcyjnosc L [H]:  0.03`
	`Pojemnosc C [F]:  6.0e-5`
	`---`
	`Czestotliwosc fr [Hz]:  118.508408`
		$f_r = \frac{1}{2\pi}\sqrt{\frac{1}{LC}-\frac{1}{(RC)^2}}$ (5)

d)	Rezystancja R [Om]: 10 Indukcyjnosc L [H]: 1 Pojemnosc C [F]: 1.0e-6 --- Czestotliwosc fr [Hz]: 159.146988	$f_r = \frac{1}{2\pi}\sqrt{\frac{1}{LC} - \left(\frac{R}{L}\right)^2}$ (6)
e)	Rezystancja R [Om]: 100 Indukcyjnosc L [H]: 0.05 Pojemnosc C [F]: 5.0e-3 --- Czestotliwosc fr [Hz]: 10.060807	$f_r = \frac{1}{2\pi\sqrt{LC}}\sqrt{1 - \frac{L}{R^2C}}$ (7)

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://www.codeblocks.org/ - Code::Blocks

5. Pytania kontrolne

1. Wyjaśnij do czego służą zmienne w programie?
2. W jaki sposób umieszcza się komentarze w kodzie programu?
3. Scharakteryzuj typy zmiennych występujące w języku C.
4. Podaj zasady obowiązujące przy tworzeniu nazw zmiennych.

5. Scharakteryzuj operatory arytmetyczne w języku C oraz sposób tworzenia i obliczania wyrażeń arytmetycznych.
6. Wyjaśnij pojęcie rzutowania oraz podaj przykłady jego zastosowania.
7. Opisz sposoby formatowania łańcucha wyjściowego w funkcji **printf()**.
8. Opisz zasadę działania funkcji **scanf()**.

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.