Header math.h

Header	Opis zawartych funkcji	Przykładowe funkcje
math.h	Stałe i funkcje matematyczne	**Stałe matematyczne:** M_E, M_LOG2E, M_LOG10E, M_LN2, M_LN10, M_PI, M_PI_2, M_PI_4, M_1_PI, M_2_PI, M_2_SQRTPI, M_SQRT2, M_SQRT_2 **Funkcje:** round (), floor(), ceil(), trunc(), fmod() sin(), cos(), tan(), asin(), acos(), atan(), atan2() sinh(), cosh(), tanh(), asinh(), acosh(), atanh() sqrt(), cbrt(), exp(), exp2(), pow(), hypot() log(), log10(), log2() fdim(), fmin(), fmax() abs(), fabs()

#include<math.h>

Predefiniowane stałe:

```
M_E         e           = 2.71828182845904523536
M_LOG2E     lb(e)       = 1.44269504088896340736
M_LOG10E    log10e      = 0.43429448190325182765l
M_LN2       ln(2)       = 0.69314718055994530942
M_LN10      ln(10)      = 2.30258509299404568402
M_PI        pi          = 3.14159265358979323846
M_PI_2      pi/2        = 1.57079632679489661923
M_PI_4      pi/4        = 0.78539816339744830962
M_1_PI      1/pi        = 0.31830988618379067154
M_2_PI      2/pi        = 0.63661977236758134308
M_1_SQRTPI  1/sqrt(pi)  = 0.56418958354775628695   <-- tego już nie ma (było w 'Turbo C++')
M_2_SQRTPI  2/sqrt(pi)  = 1.12837916709551257390
M_SQRT2     sqrt(2)     = 1.41421356237309504880
M_SQRT_2    1/sqrt(2)   = 0.70710678118654752440
```

```c
// Już zdefiniowane stałe matematyczne

#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  printf("\n - - - - - - Predefiniowane stałe matematyczne - - - - - \n");

  printf("\n   Liczba Eulera (liczba Nepera) e =           %f", M_E);
  printf("\n   Logarytm binarny (podstawa 2) liczby e =    %f", M_LOG2E);
  printf("\n   Logarytm dziesiętny liczby e =              %f", M_LOG10E);
  printf("\n   Logarytm naturalny liczby 2 =               %f", M_LN2);
  printf("\n   Logarytm naturalny liczby 10 =              %f", M_LN10);
  printf("\n   Liczba pi =                                 %f", M_PI);
  printf("\n   Liczba pi/2 =                               %f", M_PI_2);
  printf("\n   Liczba pi/4 =                               %f", M_PI_4);
```

```
  printf("\n   Liczba 1/pi (odwrotność pi) =                %f", M_1_PI);
  printf("\n   Liczba 2/pi =                                %f", M_2_PI);
//  printf("\n   Liczba 1/sqrt(pi) =                           %f", M_1_SQRTPI);
  printf("\n   Liczba 2/sqrt(pi) =                          %f", M_2_SQRTPI);
  printf("\n   Pierwiastek drugiego stopnia liczby 2 =      %f", M_SQRT2);
  printf("\n   1/sqrt(2). Odwrotność pierwiastka liczby 2 = %f", M_SQRT1_2);
  printf("\n              To także sqrt(2)/2 \n");

  return 0;
}
```

Wynik działania programu:

```
 - - - - - - Predefiniowane stałe matematyczne - - - - -

   Liczba Eulera (liczba Nepera) e =            2.718282
   Logarytm binarny (podstawa 2) liczby e =     1.442695
   Logarytm dziesiętny liczby e =               0.434294
   Logarytm naturalny liczby 2 =                0.693147
   Logarytm naturalny liczby 10 =               2.302585
   Liczba pi =                                  3.141593
   Liczba pi/2 =                                1.570796
   Liczba pi/4 =                                0.785398
   Liczba 1/pi (odwrotność pi) =                0.318310
   Liczba 2/pi =                                0.636620
   Liczba 2/sqrt(pi) =                          1.128379
   Pierwiastek drugiego stopnia liczby 2 =      1.414214
   1/sqrt(2). Odwrotność pierwiastka liczby 2 = 0.707107
              To także sqrt(2)/2

--------------------------------
Process exited after 10.06 seconds with return value 0
Press any key to continue . . . _
```

Zaokrąglanie liczb zmiennoprzecinkowych do liczb całkowitych i otrzymywanie reszty z dzielenia

- **round (x)** - zaokrągla do najbliższej liczby całkowitej
- **floor (x)** - zaokrągla w dół
- **ceil (x)** - zaokrągla w górę
- **trunc (x)** - obcina część ułamkową i pozostawia całkowitą (zaokrągla w dół)
- **fmod (x, y)** - daje resztę z dzielenia x/y

Jeżeli chciałbyś zaokrąglić liczbę zmiennoprzecinkową do np. dwóch lub trzech miejsc po przecinku lub liczbę całkowitą czy zmiennoprzecinkową do np. stu lub tysiąca to pomnóż/podziel tę liczbę przez 100 lub 1000 a po zaokrągleniu odpowiednio podziel/pomnóż tę liczbę przez 100 lub 1000.

```
// Zaokrąglanie liczb do liczb całkowitych i reszta z dzielenia

#include<stdio.h>
#include<math.h>                  // ten temat
#include<locale.h>                // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki
```

```
float a1 = 4;
float a2 = 4.3;
float a3 = 4.5;
float a4 = 4.7;
float a5 = -4;
float a6 = -4.3;
float a7 = -4.5;
float a8 = -4.7;

float z = -12.345678;
float x = 8, y = 5;

printf("\n - - - - - Zaokrąglanie liczb do liczby całkowitej - - - - -");

printf("\n\n   Zaokrąglanie do najbliższej liczby całkowitej 'round(liczba)'");
printf("\n      %f   -->   %f", a1, round(a1) );
printf("\n      %f   -->   %f", a2, round(a2) );
printf("\n      %f   -->   %f", a3, round(a3) );
printf("\n      %f   -->   %f", a4, round(a4) );
printf("\n      %f   -->   %f", a5, round(a5) );
printf("\n      %f   -->   %f", a6, round(a6) );
printf("\n      %f   -->   %f", a7, round(a7) );
printf("\n      %f   -->   %f", a8, round(a8) );

printf("\n\n   Zaokrąglanie w dół 'floor(liczba)'");
printf("\n      %f   -->   %f", a1, floor(a1) );
printf("\n      %f   -->   %f", a2, floor(a2) );
printf("\n      %f   -->   %f", a3, floor(a3) );
printf("\n      %f   -->   %f", a4, floor(a4) );
printf("\n      %f   -->   %f", a5, floor(a5) );
printf("\n      %f   -->   %f", a6, floor(a6) );
printf("\n      %f   -->   %f", a7, floor(a7) );
printf("\n      %f   -->   %f", a8, floor(a8) );

printf("\n\n   Zaokrąglanie w górę 'ceil(liczba)'");
printf("\n      %f   -->   %f", a1, ceil(a1) );
printf("\n      %f   -->   %f", a2, ceil(a2) );
printf("\n      %f   -->   %f", a3, ceil(a3) );
printf("\n      %f   -->   %f", a4, ceil(a4) );
printf("\n      %f   -->   %f", a5, ceil(a5) );
printf("\n      %f   -->   %f", a6, ceil(a6) );
printf("\n      %f   -->   %f", a7, ceil(a7) );
printf("\n      %f   -->   %f", a8, ceil(a8) );

printf("\n\n   Obcinanie 'trunc(liczba)'");
printf("\n      %f   -->   %f", a1, trunc(a1) );
printf("\n      %f   -->   %f", a2, trunc(a2) );
printf("\n      %f   -->   %f", a3, trunc(a3) );
printf("\n      %f   -->   %f", a4, trunc(a4) );
printf("\n      %f   -->   %f", a5, trunc(a5) );
printf("\n      %f   -->   %f", a6, trunc(a6) );
printf("\n      %f   -->   %f", a7, trunc(a7) );
printf("\n      %f   -->   %f", a8, trunc(a8) );

printf("\n\n   Zaokrąglanie 'round(liczba)' liczby %f do 1/100", z);
printf("\n      %f   -->   ", z);
z *= 100; z = round(z); z /= 100; printf("%f", z);
```

```
  printf("\n\n - - - - - - - - - - Reszta z dzielenia - - - - - - - - - -");

  printf("\n\n   Reszta z dzielenia 'fmod(liczba_1, liczba_2)'");
  printf("\n      %f / %f   -->   %f   -->   %d", x, y, fmod(x, y), (int)fmod(x, y) );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
- - - - - Zaokrąglanie liczb do liczby całkowitej - - - - -

  Zaokrąglanie do najbliższej liczby całkowitej 'round(liczba)'
     4.000000   -->   4.000000
     4.300000   -->   4.000000
     4.500000   -->   5.000000
     4.700000   -->   5.000000
     -4.000000   -->   -4.000000
     -4.300000   -->   -4.000000
     -4.500000   -->   -5.000000
     -4.700000   -->   -5.000000

  Zaokrąglanie w dół 'floor(liczba)'
     4.000000   -->   4.000000
     4.300000   -->   4.000000
     4.500000   -->   4.000000
     4.700000   -->   4.000000
     -4.000000   -->   -4.000000
     -4.300000   -->   -5.000000
     -4.500000   -->   -5.000000
     -4.700000   -->   -5.000000

  Zaokrąglanie w górę 'ceil(liczba)'
     4.000000   -->   4.000000
     4.300000   -->   5.000000
     4.500000   -->   5.000000
     4.700000   -->   5.000000
     -4.000000   -->   -4.000000
     -4.300000   -->   -4.000000
     -4.500000   -->   -4.000000
     -4.700000   -->   -4.000000

  Obcinanie 'trunc(liczba)'
     4.000000   -->   4.000000
     4.300000   -->   4.000000
     4.500000   -->   4.000000
     4.700000   -->   4.000000
     -4.000000   -->   -4.000000
     -4.300000   -->   -4.000000
     -4.500000   -->   -4.000000
     -4.700000   -->   -4.000000

  Zaokrąglanie 'round(liczba)' liczby -12.345678 do 1/100
     -12.345678   -->   -12.350000

- - - - - - - - - - Reszta z dzielenia - - - - - - - - - -
```

```
  Reszta z dzielenia 'fmod(liczba_1, liczba_2)'
     8.000000 / 5.000000   -->   3.000000   -->   3

--------------------------------
Process exited after 10.43 seconds with return value 0
Press any key to continue . . .
```

Dodatkowo 'Turbo C++' oferuje funkcję **bcd()** stosującą 'zaokrąglanie bankowe', które zaokrągla do najbliższej cyfry parzystej. Np.:

bcd (12.335, 2) = 12.34
bcd (12.345, 2) = 12.34
bcd (12.355, 2) = 12.36

Funkcje trygonometryczne

Wartości argumentów, jeżeli dotyczą kątów, pobierane są w radianach.

Mała tabela miary kątów: stopnie <--> radiany:

0,5236 rad	30°
0,7854 rad	45°
1,0472 rad	60°
1,5708 rad	90°
2,0944 rad	120°
2,3562 rad	135°
2,6180 rad	150°
3,1416 rad	180° (π radianów)

- **sin (x)** - sinus x Dziedzina: (−∞; +∞)
- **cos (x)** - cosinus x Dziedzina: (−∞; +∞)
- **tan (x)** - tangens x Dziedzina: (−∞; +∞) \ (π/2 + kπ), gdzie k to liczba całkowita
- **asin (x)** - arcus sinus x Dziedzina: [-1;1]
- **acos (x)** - arcus cosinus x Dziedzina: [-1;1]
- **atan (x)** - arcus tangens x Dziedzina: (−∞; +∞)
- **atan2 (y, x)** - arcus tangens dla kąta, którego pierwsze ramię to nieujemna półoś OX a drugie zawierające współrzędne x i y. Wraz ze wzrostem miary kąta przyjmuje wartości od 0 do 90 (pierwsza ćwiartka), od 90 do 180 (druga ćwiartka), od -180 do -90 (trzecia ćwiartka) i od -90 do -0 (czwarta ćwiartka)
  Dziedzina: ((−∞; +∞), (−∞; +∞)) \ (0;0)

cotangens to odwrotność tangensa
ctg(x) = 1/tan(x) --> **1/tan(x)**

arcus cotangens(x) to π/2 - arcus tangens(x)
actg(x) = π/2 - atan(x) --> **M_PI_2 - atan(x)**

Funkcje cyklometryczne:
arcus sinus (arcsin) jest funkcją odwrotną do funkcji sinus rozpatrywanej na przedziale [-π/2, π/2]
arcus cosinus (arccos) jest funkcją odwrotną do funkcji cosinus rozpatrywanej na przedziale [0, π]
arcus tangens (arctg) jest funkcją odwrotną do funkcji tangens rozpatrywanej na przedziale [-π/2, π/2]
arcus cotangens (arcctg) jest funkcją odwrotną do funkcji cotangens rozpatrywanej na przedziale [0, π]

Zauważ drobne różnice w zapisach funkcji, zarówno w podręcznikach jak i składni języka 'C'.

```
// Funkcje trygonometryczne
```

```c
#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

// 1 rad = 180° / π = 57,295779513°
#define radian_na_stopnie 57.295779513

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  float k_rad = 2.3562;             // kąt w radianach
  float k_stp = k_rad * radian_na_stopnie;
  float k1 = 1;                     // wartość argumentu dla funkcji cyklometrycznych
  float x = -5, y = 5;              // dla atan2(x,y). (-5,5) to będzie kąt 135°

  printf("\n - - - - - - Funkcje trygonometryczne - - - - - - \n");

  printf("\n   sinus(%.2f rad)     = sinus(%.2f°)     = %.2f", k_rad, k_stp, sin(k_rad) );
  printf("\n   cosinus(%.2f rad)   = cosinus(%.2f°)   = %.2f", k_rad, k_stp, cos(k_rad) );
  printf("\n   tangens(%.2f rad)   = tangens(%.2f°)   = %.2f", k_rad, k_stp, tan(k_rad) );
  printf("\n   cotangens(%.2f rad) = cotangens(%.2f°) = %.2f", k_rad, k_stp, 1 / tan(k_rad) );

  printf("\n\n - - - - - -  Funkcje cyklometryczne  - - - - - - \n");

  printf("\n   arcsin(%.2f) = %.2f rad = %.2f°", k1, asin(k1), asin(k1) * radian_na_stopnie );
  printf("\n   arccos(%.2f) = %.2f rad = %.2f°", k1, acos(k1), acos(k1) * radian_na_stopnie );
  printf("\n   arctg(%.2f)  = %.2f rad = %.2f°", k1, atan(k1), atan(k1) * radian_na_stopnie );
  printf("\n   arcctg(%.2f) = %.2f rad = %.2f°", k1, M_PI_2 - atan(k1), (M_PI_2 - atan(k1)) *
radian_na_stopnie );
  printf("\n\n   arctg() gdy druga półoś kąta przechodzi przez punkt o współrzędnych x=%.1f, \
y=%.1f", x, y);
  printf("\n   arctg(%.1f,%.1f) = %.2f rad = %.2f°", x, y, atan2(y,x), atan2(y,x) *
radian_na_stopnie );

  printf("\n");
  return 0;
}
```

Uwaga: ASCII dla symbolu stopnia (°) strony kodowej 852 (polska strona kodowa) to numer 248. Gdybyś miał kłopoty z uzyskaniem symbolu ° to przyciśnij [Alt] i trzymając go przyciskaj kolejno 2 4 8 (w tym momencie uwolnij [Alt]) w bloku numerycznym (tym, po prawej stronie klawiatury a nie u jej góry). Może się okazać, że prawy [Alt] działa w tym przypadku inaczej niż lewy [Alt] - wypróbuj obydwa przyciski [Alt].

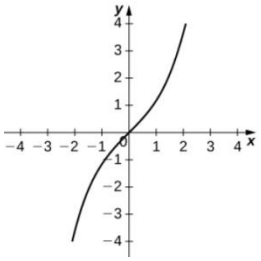
Wynik działania programu:

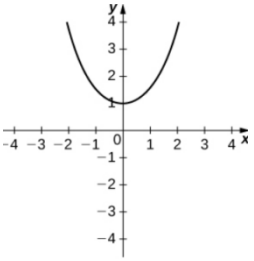
```
 - - - - - - Funkcje trygonometryczne - - - - - -

   sinus(2.36 rad)     = sinus(135.00°)     = 0.71
   cosinus(2.36 rad)   = cosinus(135.00°)   = -0.71
   tangens(2.36 rad)   = tangens(135.00°)   = -1.00
   cotangens(2.36 rad) = cotangens(135.00°) = -1.00

- - - - - -  Funkcje cyklometryczne  - - - - - -

   arcsin(1.00) = 1.57 rad = 90.00°
   arccos(1.00) = 0.00 rad = 0.00°
   arctg(1.00)  = 0.79 rad = 45.00°
```

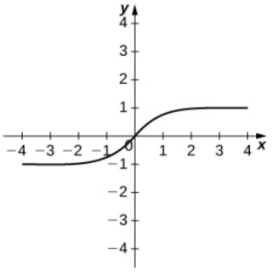
```
   arcctg(1.00) = 0.79 rad = 45.00°

   arctg() gdy druga półoś kąta przechodzi przez punkt o współrzędnych x=-5.0, y=5.0
   arctg(-5.0,5.0) = 2.36 rad = 135°

--------------------------------
Process exited after 13.08 seconds with return value 0
Press any key to continue . . . _
```

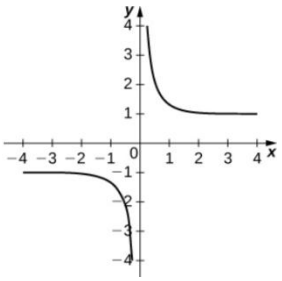
Funkcje hiperboliczne

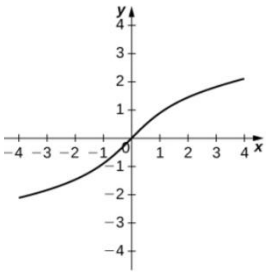
- **sinh (x)** - sinus hiperboliczny x — Dziedzina: (−∞; +∞)
- **cosh (x)** - cosinus hiperboliczny x — Dziedzina: (−∞; +∞)
- **tanh (x)** - tangens hiperboliczny x — Dziedzina: (−∞; +∞)
  cotangens hiperboliczny x — Dziedzina: (–∞,0) U (0, +∞)
- **asinh (x)** - arcus sinus hiperboliczny x — Dziedzina: (−∞; +∞)
- **acosh (x)** - arcus cosinus hiperboliczny x — Dziedzina: [1, +∞)
- **atanh (x)** - arcus tangens hiperboliczny x — Dziedzina: (-1,1)
  arcus cotangens hiperboliczny x — Dziedzina: (–∞,–1) U (1, +∞)

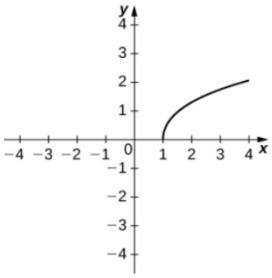

sinh(x)

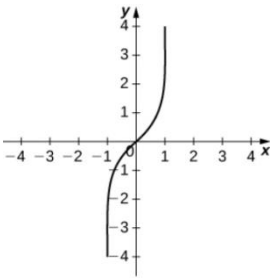

cosh(x)

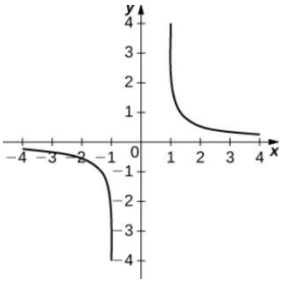

tanh(x)


coth(x)


arcsinh(x)


arccosh(x)


arctanh(x)


arccoth(x)

```c
// Funkcje hiperboliczne

#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

// 1 rad = 180° / π = 57,295779513°
#define radian_na_stopnie 57.295779513

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  float k_rad = 0.7854;           // kąt w radianach (0.7854 rad to 45°)
  float k_stp = k_rad * radian_na_stopnie;  // zamiana radianów na stopnie
```

```
  float k1 = 1.5;                 // wartość argumentu funkcji odwrotnych asinh() i acosh()
  float k2 = 0.5;                 // wartość argumentu funkcji atgh()
  float k3 = 2;                   // wartość argumentu funkcji actgh()

  printf("\n - - - - - - Funkcje hiperboliczne - - - - - - \n");

  printf("\n   sinh(%.2f rad) = sinh(%.2f°) = %.2f", k_rad, k_stp, sinh(k_rad) );
  printf("\n   cosh(%.2f rad) = cosh(%.2f°) = %.2f", k_rad, k_stp, cosh(k_rad) );
  printf("\n   tgh(%.2f rad)  = tgh(%.2f°)  = %.2f", k_rad, k_stp, tanh(k_rad) );
  printf("\n   ctgh(%.2f rad) = ctgh(%.2f°) = %.2f", k_rad, k_stp, 1 / tanh(k_rad) );

  printf("\n\n - - - - Funkcje hiperboliczne odwrotne - - - - \n");

  printf("\n   arcsinh(%.2f) = %.2f rad = %.2f°", k1, asinh(k1), asinh(k1) * radian_na_stopnie );
  printf("\n   arccosh(%.2f) = %.2f rad = %.2f°", k1, acosh(k1), acosh(k1) * radian_na_stopnie );
  printf("\n   arctgh(%.2f)  = %.2f rad = %.2f°", k2, atanh(k2), atanh(k2) * radian_na_stopnie );
  printf("\n   arcctgh(%.2f) = %.2f rad = %.2f°", k3, atanh(1/k3), atanh(1/k3) * radian_na_stopnie );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
 - - - - - - Funkcje hiperboliczne - - - - - -

   sinh(0.79 rad) = sinh(45.00°) = 0.87
   cosh(0.79 rad) = cosh(45.00°) = 1.32
   tgh(0.79 rad)  = tgh(45.00°)  = 0.66
   ctgh(0.79 rad) = ctgh(45.00°) = 1.52

 - - - - Funkcje hiperboliczne odwrotne - - - -
   arcsinh(1.50) = 1.19 rad = 68.45°
   arccosh(1.50) = 0.96 rad = 55.14°
   arctgh(0.50)  = 0.55 rad = 31.47°
   arcctgh(4.00) = 0.26 rad = 14.63°

--------------------------------
Process exited after 10.13 seconds with return value 0
Press any key to continue . . .
```

Pierwiastki i potęgi

- **sqrt (x)** - pierwiastek drugiego stopnia (kwadratowy) liczby x Dziedzina: [0; +∞)
- **cbrt (x)** - pierwiastek trzeciego stopnia (sześcienny) liczby x Dziedzina: (−∞; +∞)
- **exp (x)** - e^x Dziedzina: (−∞; +∞)
- **exp2 (x)** - 2^x Dziedzina: (−∞; +∞)
- **pow (x, y)** - x^y Dziedzina: ((−∞; +∞), (−∞; +∞))
- **hypot (x, y)** - długość przeciwprostokątnej, gdy długości przyprostokątnych to x i y
  Dziedzina: ((0; +∞), (0; +∞))

Pierwiastek n-tego stopnia to potęga o wykładniku 1/n.
Pierwiastek 4-go stopnia liczby 5 to potęga o wykładniku 0.25 --> **pow(5, 0.25)**

```
// Pierwiastki i potęgi
```

```c
#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  float x2 = 2, x3 = 3, x4 = 4, x5 = 5;

  printf("\n - - - - - - - - - - Pierwiastki i potęgi - - - - - - - - - -");

  printf("\n\n   * Pierwiastki");

  printf("\n   Pierwiastek drugiego stopnia liczby %.2f    to   %.4f", x2, sqrt(x2) );
  printf("\n   Pierwiastek trzeciego stopnia liczby %.2f   to   %.4f", x2, cbrt(x2) );
  printf("\n   Pierwiastek %.0f stopnia liczby %.2f           to   %.4f", x4, x5,
pow(x5, 1/x4) );

  printf("\n\n   * Potęgi");

  printf("\n   Liczba Nepera (e) podniesiona do potęgi %.2f   to   %.4f", x2, exp(x2)
);
  printf("\n   Liczba 2 podniesiona do potęgi %.2f      to   %.2f", x5, exp2(x5) );
  printf("\n   Liczba %.2f podniesiona do potęgi %.2f   to   %.2f", x3, x4, pow(x3,
x4) );

  printf("\n\n   * Długość przeciwprostokątnej");
  printf("\n   Długość przeciwprostokątnej, gdy przyprostokątne to %.2f i %.2f \
wynosi %.2f", x3, x4, hypot(x3, x4) );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
 - - - - - - - - - - Pierwiastki i potęgi - - - - - - - - - -

   * Pierwiastki
   Pierwiastek drugiego stopnia liczby 2.00    to   1.4142
   Pierwiastek trzeciego stopnia liczby 2.00   to   1.2599
   Pierwiastek 4 stopnia liczby 5.00           to   1.4953

   * Potęgi
   Liczba Nepera (e) podniesiona do potęgi 2.00   to   7.3891
   Liczba 2 podniesiona do potęgi 5.00      to   32.00
   Liczba 3.00 podniesiona do potęgi 4.00   to   81.00

   * Długość przeciwprostokątnej
   Długość przeciwprostokątnej, gdy przyprostokątne to 3.00 i 4.00 wynosi 5.00

--------------------------------
Process exited after 13.48 seconds with return value 0
Press any key to continue . . .
```

Logarytmy

- **log (x)** - ln(x), czyli $\log_e(x)$ Dziedzina: (0; +∞)
- **log10 (x)** - $\log_{10}(x)$ Dziedzina: (0; +∞)
- **log2 (x)** - $\log_2(x)$ to znaczy lb(x) Dziedzina: (0; +∞)

Logarytm o dowolnej podstawie:

$\log_a(b) = \log_c(b) / \log_c(a)$ --> **log10(b) / log10(a)**

lub

$\log_a(b) = \ln(b) / \ln(a)$ --> **log(b) / log(a)**

```c
// Logarytmy

#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  float x100 = 100, x1000 = 1000, x1024 = 1024, x81 = 81;
  float a = 3;

  printf("\n\n - - - - - - - - - - - - Logarytmy - - - - - - - - - - - -");

  printf("\n   Logarytm naturalny liczby %.2f, ln(%.2f) to %.4f", x1000, x1000,
log(x1000) );
  printf("\n   Logarytm dziesiętny liczby %.2f, lg(%.2f) to %.4f", x1000, x1000,
log10(x1000) );
  printf("\n   Logarytm binarny liczby %.2f, lb(%.2f) to %.4f", x1024, x1024,
log2(x1024) );
  printf("\n   Logarytm przy podstawie %.2f liczby %.2f to %.4f", a, x81, log10(x81) /
log10(a) );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
 - - - - - - - - - - - - Logarytmy - - - - - - - - - - - -

   Logarytm naturalny liczby 1000.00, ln(1000.00) to 6.9078
   Logarytm dziesiętny liczby 1000.00, lg(1000.00) to 3.0000
   Logarytm binarny liczby 1024.00, lb(1024.00) to 10.0000
   Logarytm przy podstawie 3.00 liczby 81.00 to 4.0000

--------------------------------
Process exited after 0.5 seconds with return value 0
Press any key to continue . . .
```

Sprawdzenie:

$\ln(1000) = 6.9078$ <--> $e^{6.9078} = 2.718282^{6.9078} \approx 1000.05 \approx 1000$

$\lg(1000) = 3$ <--> $10^3 = 1000$

lb(1024) = 10 <--> 2^{10} = 1024
$\log_3(81)$ = 4 <--> 3^4 = 81

Porównanie dwóch liczb

- **fdim (x, y)** - daje dodatnią różnicę pomiędzy x i y. Jeżeli x-y ≤ 0, daje 0
- **fmin (x, y)** - daje mniejszą liczbę z dwóch liczb zmiennoprzecinkowych, traktując NAN jak brak danych
- **fmax (x, y)** - daje większą liczbę z dwóch liczb zmiennoprzecinkowych, traktując NAN jak brak danych

Wszystkie przypadki - Dziedzina: ((−∞; +∞), (−∞; +∞))

```c
// Porównanie dwóch liczb

#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  float a = 100.1111, b = 500.5555, c = -700.7777;

  printf("\n - - - - - - - - - - Porównanie dwóch liczb - - - - - - - - - -\n");

  printf("\n   Dodatnia różnica dwóch liczb: %.2f i %.2f to %.2f", a, b, fdim(a, b) );
  printf("\n   Dodatnia różnica dwóch liczb: %.2f i %.2f to %.2f", b, a, fdim(b, a) );
  printf("\n   Dodatnia różnica dwóch liczb: %.2f i %.2f to %.2f", a, c, fdim(a, c) );
  printf("\n");
  printf("\n   Z dwóch liczb: %.2f i %.2f najmniejszą jest %.2f", a, b, fmin(a, b) );
  printf("\n   Z dwóch liczb: %.2f i %.2f największą jest  %.2f", a, b, fmax(a, b) );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
 - - - - - - - - - - Porównanie dwóch liczb - - - - - - - - - -

   Dodatnia różnica dwóch liczb: 100.11 i 500.56 to 0.00
   Dodatnia różnica dwóch liczb: 500.55 i 100.11 to 400.44
   Dodatnia różnica dwóch liczb: 100.11 i -700.78 to 800.89

   Z dwóch liczb: 100.11 i 500.56 najmniejszą jest 100.11
   Z dwóch liczb: 100.11 i 500.56 największą jest  500.56

--------------------------------
Process exited after 10.09 seconds with return value 0
Press any key to continue . . .
```

Wartość bezwzględna liczby

- **abs (x)** - wartość absolutna liczby całkowitej
- **fabs (x)** - wartość absolutna liczby zmiennoprzecinkowej

Wszystkie przypadki - Dziedzina: (−∞; +∞)

```c
// Wartość bezwzględna liczby

#include<stdio.h>
#include<math.h>                    // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main(void)
{
  setlocale(LC_CTYPE, "Polish");  //polskie znaki

  int   i = 100, j = -100, k = 0;
  float f = 10.123, g = -10.123;

  printf("\n - - - - - - - Wartość bezwzględna liczby - - - - - - -\n");

  printf("\n   Wartość bezwzględna %d to %d", i, abs(i) );
  printf("\n   Wartość bezwzględna %d to %d", j, abs(j) );
  printf("\n   Wartość bezwzględna %d to %d", k, abs(k) );
  printf("\n");
  printf("\n   Wartość bezwzględna %.2f to %.2f", f, fabs(f) );
  printf("\n   Wartość bezwzględna %.2f to %.2f", g, fabs(g) );

  printf("\n");
  return 0;
}
```

Wynik działania programu:

```
 - - - - - - - Wartość bezwzględna liczby - - - - - - -

   Wartość bezwzględna 100 to 100
   Wartość bezwzględna -100 to 100
   Wartość bezwzględna 0 to 0

   Wartość bezwzględna 10.12 to 10.12
   Wartość bezwzględna -10.12 to 10.12

--------------------------------
Process exited after 13.49 seconds with return value 0
Press any key to continue . . .
```