# Zadanie B
# Root finder

Metoda bisekcji w teorii zawsze znajduje zero funkcji jeżeli rozpoczniemy od dwóch punktów, nawet bardzo oddalonych, w których funkcja ma przeciwne znaki. Niestety robi to wolno. Z drugiej strony metoda siecznych jest zbieżna szybciej ale tylko lokalnie.

Zadanie polega na napisaniu w C++ funkcji o sygnaturze

```
double findZero(
    double (*f)(double),  // funkcja której zera szukamy w [a, b]
    double a,             // lewy koniec przedziału
    double b,             // prawy koniec przedziału
    int M,                // maksymalna dozwolona liczba wywołań
funkcji f
    double eps,           // spodziewana dokładność zera
    double delta          // wystarczający błąd bezwzględny wyniku
);
```

która połączy te dwie metody wyznaczania zer funkcji nieliniowych tak aby uzyskać metodę o dobrych cechach globalnych i lokalnych.
Dla przekazanej ciągłej funkcji f i przedziału [a,b] funkcja ma zwrócić przybliżenie x0 zera funkcji f.

- Jeżeli na wejściu f(a)f(b) ≤ 0 to x0 powinno należeć do przedziału [a,b], w przeciwnym wypadku szukamy dowolnego zera rozpoczynając iterację metody siecznych od x0 = a i x1 = b. Każda z testowych funkcji będzie miała przynajmniej jedno zero.
- Funkcję f można **wywołać co najwyżej M razy**. Przekroczenie tego limitu będzie zgłaszane przez błąd wykonania programu.
- Wynik x0 zostanie zaakceptowany jeżeli |f(x0)| ≤ eps lub |x -x0| ≤ delta dla pewnego x - zera funkcji f.

Na BaCę należy przesłać plik *source.cpp* zawierający funkcję findZero i ewentualne funkcje pomocnicze ale nie zawierający funkcji main.

## Przykład

```
#include "source.cpp"
#include <iostream>
#include <cmath>
using namespace std;

double wielomian(double x){     return (((x-6)*x+11)*x)-6;      }
double wielomianSinExp(double x)
{       return ((((x-6)*x+11)*x)-4 + sin(15*x))*exp(-x*x);      }
double kwadrat(double x){       return (x*x-2);  }
double kwadrat100(double x){  return 1e100*(x*x-2);     }
double kwadrat_10(double x){    return 1e-10*(x*x-2);   }

int main(){
        cout.precision(17);
        cout << findZero(wielomian, 0, 4, 20, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, 0, 40, 20, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, 1, 2, 2, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, -150, 1.9, 20, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, 1.5, 2.99, 20, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, 2.01, 40, 20, 1e-15, 1e-14) << endl;
        cout << findZero(wielomian, 1.5, 6, 20, 1e-15, 1e-14) << endl;

        cout << findZero(wielomianSinExp, -1, 3, 60, 1e-60, 1e-14) << endl;
        cout << findZero(wielomianSinExp, -3, 3, 60, 1e-160, 1e-14) << endl;

        cout << findZero(kwadrat, 0, 4, 15, 1e-11, 1e-14) << endl;
        cout << findZero(kwadrat100, 0, 4, 15, 1e-11, 1e-14) << endl;
        cout << findZero(kwadrat_10, 0, 4, 10, 1e-10, 1e-14) << endl;
        cout << findZero(kwadrat_10, 0, 4, 15, 1e-160, 1e-14) << endl;
        return 0;
}
```

```
Spodziewane alternatywy wyjścia:
1 lub 2 lub 3
1 lub 2 lub 3
1 lub 2
1
2
3
1 lub 2 lub 3
0.43636925909804245
0.43636925909804245
1.414213562373095
1.414213562373095
każdy punkt z przedziału [1, 1.73205]
1.414213562373095
```