Zadanie 0 – Metoda Adamsa-Bashforda (1pkt)

Uzupełnij poniższy szablon funkcji, rozwiązującej równanie różniczkowe metodą Adamsa-Bashforda 2 rzędu w punkcie x:

```
template <class T>
double AdamsBashford(T f, double x, double x0, double y0, double step)
```

Pomocne wzory:

$$y_{n+2} = y_{n+1} + h\left(\frac{3}{2}f(x_{n+1}, y_{n+1}) - \frac{1}{2}f(x_n, y_n)\right) \tag{1}$$

gdzie h jest krokiem metody. x_0 oraz y_0 są podane jako argumenty szablonu. Na ich podstawie wylicz x_1 i y_1 metodą Eulera, a kolejne punkty z powyższego wzoru. Rozwiązanie ma działać niezależnie od tego czy $x > x0$ czy $x < x0$ lub $x == x0$ oraz niezależnie od znaku podanego kroku. Działanie ma się zakończyć, gdy kolejna iteracja oznaczałaby "przeskoczenie" wartości x dla której liczymy (tak samo jak w poprzedniej serii).

Szablon:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>
#include <string>
#include <functional>
using namespace std;

//pomocna funkcja
template <typename T> int sgn(T val)
{
        if (val > 0)
                return 1;
        else if (val < 0)
                return -1;
        else
                return 0;
}

template <class T>
double AdamsBashford(T f, double x, double x0, double y0, double step);

double eq(double x, double y)
{
        return y;
}

double poly(double x, double y)
{
        return 3 * x * x;
}

double bernoulli(double x, double y)
```

```
{
        if (x == 0)
                return NAN;
        else
                return x * y * y - y / x;
}

//jaki fajny szablon!
template <typename T>
using funcPointer = double (*)(T, double, double, double, double);

int main()
{
        vector<double> step = { 1., 0.25, 0.1, 0.05, 0.025, 0.0125 };

        cout <<"Adams-Bashford method:" << setprecision(6) << endl;
        cout << "y'=y and y(0)=1 -> y(3)=exp(3)=20,0855"
                << endl;
        for (auto s : step)
        {
                double res = AdamsBashford(eq, 3, 0, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res - exp(3)) << endl;
        }
        cout << endl;

        cout << "y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27"
                << endl;
        for (auto s : step)
        {
                double res = AdamsBashford(poly, -3, 1, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res + 27) << endl;
        }
        cout << endl;

        cout << "y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111"
                << endl;
        for (auto s : step)
        {
                double res = AdamsBashford(bernoulli, 3, 1, -1, -1 * s);
                cout << "h=" << s << " y=" <<
                        res << " err=" << abs(res + 0.111) << endl;
        }
        cout << endl;

}
```

Przykładowy output:

Adams-Bashford method:
y'=y and y(0)=1 -> y(3)=exp(3)=20,0855
h=1 y=20.5 err=0.414463
h=0.25 y=22.9932 err=2.90771
h=0.1 y=21.7397 err=1.65416
h=0.05 y=22.0768 err=1.99125
h=0.025 y=21.0861 err=1.00053
h=0.0125 y=20.3311 err=0.245559

y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27
h=1 y=-56 err=29
h=0.25 y=-33.875 err=6.875
h=0.1 y=-29.72 err=2.72
h=0.05 y=-29.7731 err=2.77306
h=0.025 y=-28.3682 err=1.3682
h=0.0125 y=-27.3378 err=0.337812

y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111
h=1 y=48.4062 err=48.5172
h=0.25 y=-0.0892404 err=0.0217596
h=0.1 y=-0.103937 err=0.00706327
h=0.05 y=-0.104082 err=0.00691848
h=0.025 y=-0.107509 err=0.00349125
h=0.0125 y=-0.110194 err=0.000805813

Zadanie 1 – Metoda Adamsa-Moultona (1pkt)

Uzupełnij poniższy szablon funkcji, rozwiązującej równanie różniczkowe metodą Adamsa-Moultona 2 rzędu w punkcie x:

```
template <class T>
double AdamsMoulton(T f, double x, double x0, double y0, double step)
```

Pomocne wzory:

$$y_{n+2} = y_{n+1} + \frac{h}{2}\left(f(x_{n+1}, y_{n+1}) + f(x_n, y_n)\right) \tag{2}$$

gdzie h jest krokiem metody. x_0 oraz y_0 są podane jako argumenty szablonu. Na ich podstawie wylicz x_1 i y_1 metodą Eulera, a kolejne punkty z powyższego wzoru. Rozwiązanie ma działać niezależnie od tego czy $x > x0$ czy $x < x0$ lub $x == x0$ oraz niezależnie od znaku podanego kroku. Działanie ma się zakończyć, gdy kolejna iteracja oznaczałaby "przeskoczenie" wartości x dla której liczymy (tak samo jak w poprzedniej serii).

Szablon:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>
#include <string>
#include <functional>
using namespace std;
```

```cpp
//pomocna funkcja
template <typename T> int sgn(T val)
{
        if (val > 0)
                return 1;
        else if (val < 0)
                return -1;
        else
                return 0;
}

template <class T>
double AdamsMoulton(T f, double x, double x0, double y0, double step);

double eq(double x, double y)
{
        return y;
}

double poly(double x, double y)
{
        return 3 * x * x;
}

double bernoulli(double x, double y)
{
        if (x == 0)
                return NAN;
        else
                return x * y * y - y / x;
}

//jaki fajny szablon!
template <typename T>
using funcPointer = double (*)(T, double, double, double, double);

int main()
{
        vector<double> step = { 1., 0.25, 0.1, 0.05, 0.025, 0.0125 };

        cout <<"Adams-Moulton method:" << setprecision(6) << endl;
        cout << "y'=y and y(0)=1 -> y(3)=exp(3)=20,0855"
                << endl;
        for (auto s : step)
        {
                double res = AdamsMoulton(eq, 3, 0, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res - exp(3)) << endl;
        }
        cout << endl;
```

```
        cout << "y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27"
                << endl;
        for (auto s : step)
        {
                double res = AdamsMoulton(poly, -3, 1, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res + 27) << endl;
        }
        cout << endl;

        cout << "y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111"
                << endl;
        for (auto s : step)
        {
                double res = AdamsMoulton(bernoulli, 3, 1, -1, 1 * s);
                cout << "h=" << s << " y=" <<
                        res << " err=" << abs(res + 0.111) << endl;
        }
        cout << endl;

}
```

Przykładowy output:

Adams-Moulton method:
y'=y and y(0)=1 -> y(3)=exp(3)=20,0855
h=1 y=12.5 err=7.58554
h=0.25 y=14.7925 err=5.293
h=0.1 y=17.1015 err=2.98403
h=0.05 y=19.2673 err=0.818197
h=0.025 y=19.6331 err=0.452438
h=0.0125 y=19.6038 err=0.481748

y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27
h=1 y=-32 err=5
h=0.25 y=-27.875 err=0.875
h=0.1 y=-27.32 err=0.32
h=0.05 y=-28.5277 err=1.52769
h=0.025 y=-27.7569 err=0.756898
h=0.0125 y=-27.0378 err=0.0378125

y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111
h=1 y=6.80208 err=6.91308
h=0.25 y=-0.0415575 err=0.0694425
h=0.1 y=-0.0809215 err=0.0300785
h=0.05 y=-0.0925877 err=0.0184123
h=0.025 y=-0.101583 err=0.00941692
h=0.0125 y=-0.107161 err=0.0038393

Zadanie 2 – Metoda Rungego-Kutty (1pkt)

Uzupełnij poniższy szablon funkcji, rozwiązującej równanie różniczkowe metodą Rungego-Kutty 4 rzędu w punkcie x:

```
template <class T>
double RungeKutta(T f, double x, double x0, double y0, double step)
```

Pomocne wzory:

$$k_1 = f(x_n, y_n) \tag{3}$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \tag{4}$$

$$k_3 = f\left(x_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \tag{5}$$

$$k_4 = f(x_n + h, y_n + h \cdot k_3) \tag{6}$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{7}$$

$$x_{n+1} = x_n + h \tag{8}$$

gdzie h jest krokiem metody. x_0 oraz y_0 są podane jako argumenty szablonu. Rozwiązanie ma działać niezależnie od tego czy $x > x0$ czy $x < x0$ lub $x == x0$ oraz niezależnie od znaku podanego kroku. Działanie ma się zakończyć, gdy kolejna iteracja oznaczałaby "przeskoczenie" wartości x dla której liczymy (tak samo jak w poprzedniej serii).

Szablon:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>
#include <string>
#include <functional>
using namespace std;

//pomocna funkcja
template <typename T> int sgn(T val)
{
        if (val > 0)
                return 1;
        else if (val < 0)
                return -1;
        else
                return 0;
}

template <class T>
double RungeKutta(T f, double x, double x0, double y0, double step);

double eq(double x, double y)
{
```

```
        return y;
}

double foo(double x, double y)
{
        return x+y;
}

double poly(double x, double y)
{
        return 3 * x * x;
}

double bernoulli(double x, double y)
{
        if (x == 0)
                return NAN;
        else
                return x * y * y - y / x;
}

//jaki fajny szablon!
template <typename T>
using funcPointer = double (*)(T, double, double, double, double);

int main()
{
        vector<double> step = { 1., 0.25, 0.1, 0.05, 0.025, 0.0125, 0.0001 };

        cout <<"RungeKutta method:" << setprecision(6) << endl;
        cout << "y'=y and y(0)=1 -> y(3)=exp(3)=20,0855"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta(eq, 3, 0, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res - exp(3)) << endl;
        }
        cout << endl;

        cout << "y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta(poly, -3, 1, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res + 27) << endl;
        }
        cout << endl;
```

```
        cout << "y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta(bernoulli, 3, 1, -1, 1 * s);
                cout << "h=" << s << " y=" <<
                        res << " err=" << abs(res + 0.111) << endl;
        }
        cout << endl;

        cout << "y'=x+y and y(0)=0 -> y(-1)=1/e=0.3678"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta(foo, -1, 0, 0, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res-0.3679) << endl;
        }
        cout << endl;

}
```

Przykładowy output:

RungeKutta method:
y'=y and y(0)=1 -> y(3)=exp(3)=20,0855
h=1 y=19.8658 err=0.219724
h=0.25 y=20.0839 err=0.00159354
h=0.1 y=20.0855 err=4.62035e-05
h=0.05 y=21.1153 err=1.0298
h=0.025 y=20.594 err=0.508468
h=0.0125 y=20.0855 err=1.21322e-08
h=0.0001 y=20.0855 err=4.61853e-14

y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27
h=1 y=-27 err=0
h=0.25 y=-27 err=0
h=0.1 y=-27 err=1.42109e-14
h=0.05 y=-28.3726 err=1.37263
h=0.025 y=-27.6806 err=0.680641
h=0.0125 y=-27 err=3.19744e-14
h=0.0001 y=-27 err=9.74154e-12

y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111
h=1 y=1.30659 err=1.41759
h=0.25 y=-0.111143 err=0.00014268
h=0.1 y=-0.111112 err=0.000111918
h=0.05 y=-0.107498 err=0.00350197
h=0.025 y=-0.109282 err=0.00171784
h=0.0125 y=-0.111111 err=0.000111111
h=0.0001 y=-0.111111 err=0.000111111

y'=x+y and y(0)=0 -> y(-1)=1/e=0.3678
h=1 y=0.375 err=0.0071
h=0.25 y=0.367894 err=5.80059e-06
h=0.1 y=0.432871 err=0.0649714
h=0.05 y=0.367879 err=2.05389e-05
h=0.025 y=0.367879 err=2.05576e-05
h=0.0125 y=0.37581 err=0.00790957
h=0.0001 y=0.367943 err=4.26551e-05

Zadanie 3 – Metoda Rungego-Kutty 3/8 (2pkt)

Uzupełnij poniższy szablon funkcji, rozwiązującej równanie różniczkowe metodą Rungego-Kutty 3/8 w punkcie x. Metoda ta jest modyfikacją MK4.

```
template <class T>
double RungeKutta3over8(T f, double x, double x0, double y0, double step)
```

Pomocne wzory:

$$k_1 = hf\left(x_n, y_n\right) \tag{9}$$
$$k_2 = hf\left(x_n + \frac{h}{3}, y_n + h\frac{k_1}{3}\right) \tag{10}$$
$$k_3 = hf\left(x_n + \frac{2h}{3}, y_n - h\frac{k_2}{3}\right) \tag{11}$$
$$k_4 = hf\left(x_n + h, y_n + k_1 - k_2 + k_3\right) \tag{12}$$
$$y_{n+1} = y_n + \frac{1}{8}\left(k_1 + 3k_2 + 3k_3 + k_4\right) \tag{13}$$
$$x_{n+1} = x_n + h \tag{14}$$

gdzie h jest krokiem metody. x_0 oraz y_0 są podane jako argumenty szablonu. Rozwiązanie ma działać niezależnie od tego czy $x > x0$ czy $x < x0$ lub $x == x0$ oraz niezależnie od znaku podanego kroku. Działanie ma się zakończyć, gdy kolejna iteracja oznaczałaby "przeskoczenie" wartości x dla której liczymy (tak samo jak w poprzedniej serii).

Szablon:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>
#include <string>
#include <functional>
using namespace std;

//pomocna funkcja
template <typename T> int sgn(T val)
{
        if (val > 0)
                return 1;
        else if (val < 0)
                return -1;
```

```
        else
                return 0;
}

template <class T>
double RungeKutta3over8(T f, double x, double x0, double y0, double step);

double eq(double x, double y)
{
        return y;
}

double foo(double x, double y)
{
        return x+y;
}

double poly(double x, double y)
{
        return 3 * x * x;
}

double bernoulli(double x, double y)
{
        if (x == 0)
                return NAN;
        else
                return x * y * y - y / x;
}

//jaki fajny szablon!
template <typename T>
using funcPointer = double (*)(T, double, double, double, double);

int main()
{
        vector<double> step = { 1., 0.25, 0.1, 0.05, 0.025, 0.0125, 0.0001 };

        cout <<"RungeKutta 3/8 method:" << setprecision(6) << endl;
        cout << "y'=y and y(0)=1 -> y(3)=exp(3)=20,0855"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta3over8(eq, 3, 0, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res - exp(3)) << endl;
        }
        cout << endl;

        cout << "y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27"
```

```
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta3over8(poly, -3, 1, 1, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res + 27) << endl;
        }
        cout << endl;

        cout << "y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta3over8(bernoulli, 3, 1, -1, 1 * s);
                cout << "h=" << s << " y=" <<
                        res << " err=" << abs(res + 0.111) << endl;
        }
        cout << endl;

        cout << "y'=x+y and y(0)=0 -> y(-1)=1/e=0.3678"
                << endl;
        for (auto s : step)
        {
                double res = RungeKutta3over8(foo, -1, 0, 0, s);
                cout << "h=" << s << " y=" << res
                        << " err=" << abs(res-0.3679) << endl;
        }
        cout << endl;

}
```

Przykładowy output:

RungeKutta 3/8 method:
y'=y and y(0)=1 -> y(3)=exp(3)=20,0855
h=1 y=19.8658 err=0.219724
h=0.25 y=20.0839 err=0.00159354
h=0.1 y=20.0855 err=4.62035e-05
h=0.05 y=21.1153 err=1.0298
h=0.025 y=20.594 err=0.508468
h=0.0125 y=20.0855 err=1.21322e-08
h=0.0001 y=20.0855 err=4.61853e-14

y'=3x^2 and y(1)=1 -> y(-3)=(-3)^3=-27
h=1 y=-27 err=0
h=0.25 y=-27 err=0
h=0.1 y=-27 err=2.13163e-14
h=0.05 y=-28.3726 err=1.37263
h=0.025 y=-27.6806 err=0.680641
h=0.0125 y=-27 err=3.19744e-14
h=0.0001 y=-27 err=9.74509e-12

y'=xy^2-y/x and y(1)=-1 -> y(3)=-1/[(-3)^2]=-0.1111
h=1 y=6.17365e+11 err=6.17365e+11
h=0.25 y=-0.111089 err=8.91126e-05
h=0.1 y=-0.111112 err=0.000111549
h=0.05 y=-0.107498 err=0.00350198
h=0.025 y=-0.109282 err=0.00171784
h=0.0125 y=-0.111111 err=0.000111111
h=0.0001 y=-0.111111 err=0.000111111

y'=x+y and y(0)=0 -> y(-1)=1/e=0.3678
h=1 y=0.375 err=0.0071
h=0.25 y=0.367894 err=5.80059e-06
h=0.1 y=0.432871 err=0.0649714
h=0.05 y=0.367879 err=2.05389e-05
h=0.025 y=0.367879 err=2.05576e-05
h=0.0125 y=0.37581 err=0.00790957
h=0.0001 y=0.367943 err=4.26551e-05