Zadanie 0 – Równania różniczkowe (3pkt)

Uzupełnij szablony funkcji do rozwiązywania równań różniczkowych postaci

$$y' = f(x, y) \tag{1}$$

metodami Eulera, midpoint i Heuna.

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

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

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

Wyjaśnienie argumentów:

1. T f – wskaźnik na funkcję przyjmującą dwa argumenty liczbowe i zwracającą liczbę (patrz szablon programu).
2. double x – argument dla którego chcemy policzyć rozwiązanie, tzn. funkcja ma zwrócić $y(x)$.
3. double x0, double y0 – warunki początkowe dla równania.
4. double step – krok iteracji po iksach.

Możliwe są trzy scenariusze:

1. $x > x0$ wtedy zwiększamy x0,
2. $x < x0$ wtedy zmniejszamy x0,
3. $x == x0$ wtedy znamy od razu wynik.

Szablony mają radzić sobie we wszystkich 3 przypadkach. Szablony mają nie sugerować się znakiem argumentu *step*. Obliczenia mają się zakończyć w momencie, gdy iterując "przeskoczymy" szukaną wartość x,[1] np. gdy $x0 = 0, x = 1.3, \text{step} = 0.5$ to wartość zwrócona będzie $y(1.0)$, bo 1.5 jest już za x. Można to zrobić patrząc czy zmienia się znak różnicy $x - x0$.

Wykorzystaj poniższy szablon:

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

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

[1] Sprytniejsze rozwiązania na zakończenie obliczeń są mile widziane, ale podane jest wystarczające.

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

template <class T>
double heun(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};
        vector<string> names = {"EULER", "MIDPOINT", "HEUN"};
        //jaki fajny wektor!
        vector<funcPointer<double (*)(double, double)> > methods
         = {eulerMethod, midPoint, heun};
        for(int ii=0; ii<methods.size(); ii++)
        {
                cout << names[ii] << " METHOD:" << setprecision(6) << endl;
                cout << "y'=y and y(0)=1 -> y(4)=exp(4)=54.5982"
                 << endl;
                for(auto s : step)
                {
                        double res =  methods[ii](eq, 4, 0, 1, s);
                        cout << "h=" << s << " y=" << res
                         << " err=" << abs(res - exp(4)) << endl;
                }
                cout << endl;

                cout << "y'=3x^2 and y(1)=1 -> y(-2)=(-2)^3=-8"
                 << endl;
```

```
                for(auto s : step)
                {
                        double res =  methods[ii](poly, -2, 1, 1, s);
                        cout << "h=" << s << " y=" << res
                         << " err=" << abs(res + 8) << endl;
                }
                cout << endl;

                cout << "y'=xy^2-y/x and y(1)=-1 -> y(2)=-1/[(-2)^2]=-0.25"
                 << endl;
                for(auto s : step)
                {
                        double res =  methods[ii](bernoulli, 2, 1, -1, 1*s);
                        cout << "h=" << s << " y=" <<
                        res << " err=" << abs(res + 0.125) << endl;
                }
                cout << endl;
        }
}
```

Przykładowy output:

EULER METHOD:
y'=y and y(0)=1 -> y(4)=exp(4)=54.5982
h=1 y=16 err=38.5982
h=0.25 y=35.5271 err=19.071
h=0.1 y=45.2593 err=9.33889
h=0.05 y=52.0395 err=2.55864
h=0.025 y=53.2773 err=1.32084
h=0.0125 y=53.2611 err=1.33704

y'=3x^2 and y(1)=1 -> y(-2)=(-2)^3=-8
h=1 y=-5 err=3
h=0.25 y=-6.96875 err=1.03125
h=0.1 y=-7.565 err=0.435
h=0.05 y=-7.77875 err=0.22125
h=0.025 y=-8.18844 err=0.188437
h=0.0125 y=-8.09398 err=0.0939844

y'=xy^2-y/x and y(1)=-1 -> y(2)=-1/[(-2)^2]=-0.25
h=1 y=1 err=1.125
h=0.25 y=-0.173591 err=0.0485909
h=0.1 y=-0.222524 err=0.0975236
h=0.05 y=-0.236649 err=0.111649
h=0.025 y=-0.23741 err=0.11241
h=0.0125 y=-0.243665 err=0.118665

MIDPOINT METHOD:
y'=y and y(0)=1 -> y(4)=exp(4)=54.5982
h=1 y=39.0625 err=15.5357
h=0.25 y=52.7402 err=1.85792

h=0.1 y=54.2614 err=0.336734
h=0.05 y=57.3042 err=2.70608
h=0.025 y=55.9573 err=1.35913
h=0.0125 y=54.5925 err=0.00563396

y'=3x^2 and y(1)=1 -> y(-2)=(-2)^3=-8
h=1 y=-7.25 err=0.75
h=0.25 y=-7.95312 err=0.046875
h=0.1 y=-7.9925 err=0.0075
h=0.05 y=-7.99813 err=0.001875
h=0.025 y=-8.30329 err=0.303293
h=0.0125 y=-8.15082 err=0.150822

y'=xy^2-y/x and y(1)=-1 -> y(2)=-1/[(-2)^2]=-0.25
h=1 y=-1 err=0.875
h=0.25 y=-0.27137 err=0.14637
h=0.1 y=-0.252463 err=0.127463
h=0.05 y=-0.250558 err=0.125558
h=0.025 y=-0.243995 err=0.118995
h=0.0125 y=-0.246936 err=0.121936

HEUN METHOD:
y'=y and y(0)=1 -> y(4)=exp(4)=54.5982
h=1 y=39.0625 err=15.5357
h=0.25 y=52.7402 err=1.85792
h=0.1 y=54.2614 err=0.336734
h=0.05 y=57.3042 err=2.70608
h=0.025 y=55.9573 err=1.35913
h=0.0125 y=54.5925 err=0.00563396

y'=3x^2 and y(1)=1 -> y(-2)=(-2)^3=-8
h=1 y=-9.5 err=1.5
h=0.25 y=-8.09375 err=0.09375
h=0.1 y=-8.015 err=0.015
h=0.05 y=-8.00375 err=0.00375
h=0.025 y=-8.30471 err=0.304711
h=0.0125 y=-8.15117 err=0.151175

y'=xy^2-y/x and y(1)=-1 -> y(2)=-1/[(-2)^2]=-0.25
h=1 y=0.75 err=0.875
h=0.25 y=-0.263116 err=0.138116
h=0.1 y=-0.251785 err=0.126785
h=0.05 y=-0.250418 err=0.125418
h=0.025 y=-0.243964 err=0.118964
h=0.0125 y=-0.246929 err=0.121929

1 Zadanie 1 – Polimorfizm i fabryka (2pkt)

1.1 Polimorfizm i metody wirtualne

1. Napisz definicję klasy o nazwie "Owoc". Klasa ta ma być **abstrakcyjna** i posiadać jedną publiczną metodę o nazwie "powitanie". Powitanie nie przyjmuje argumentów ani nie zwraca żadnej wartości, jedyne co robi, to wypisuje na *std* :: *cout* napis "Jestem sobie pysznym owocem!". Powitanie ma być metodą **wirtualną!** Nie musisz definiować innych pól i konstruktorów dla tej klasy.

2. Stwórz trzy klasy dziedziczące po klasie Owoc ze specyfikatorem dostępu "public". Klasy noszą nazwy: "Jablko", "Gruszka" i "Banan". Dla każdej z klas zdefiniuj własną publiczną metodę o identycznej nazwie i sygnaturze jak *Owoc* :: *powitanie*. Nie musisz definiować żadnych innych składowych.

3. Przetestuj działanie programu na mainie z poniższego szablonu

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

int main()
{
        //wektor wskaznikow na klase abstrakcyjna?
        vector<Owoc*> v;
        v.push_back(new Jablko);
        v.push_back(new Gruszka);
        v.push_back(new Banan);
        for(auto owoc : v)
                owoc->powitanie();
        //dostep do metody w klasie abstrakcyjnej
        v[1]->Owoc::powitanie();
        //zawsze pamietamy o zwolnieniu pamieci
        for(int ii=v.size(); ii>0; ii--)
                delete v[ii];
        return 0;
}
```

Output:
Jestem sobie pysznym jablkiem!
Jestem sobie pyszna gruszka!
Jestem sobie pysznym bananem!
Jestem sobie pysznym owocem!

1.2 Fabryka

Nie ma sensu wymyślać koła od nowa, dlatego w programowaniu używa się *wzorców projektowych*. Wzorce projektowe to gotowe rozwiązania często pojawiających się problemów, które są niezależne od używanego języka (ale przybierają zależne od języka implementacje). Przykładem wzorca projektowego, wystepującego w wielu odmianach jest fabryka.
W pierwszej części zadania stworzyliśmy 3 klasy pochodne opisujące owoce. Łatwo sobie wyobrazić, że takich klas moglibyśmy mieć o wiele więcej. Nie chcemy za każdym razem tworzyć obiektów danej klasy,

wolelibyśmy mieć jakiś interfejs, który w zależności od podanego parametru zwróci nam to co trzeba. Idealnie by było, gdyby w trakcie działania programu można było zdecydować jakie obiekty będą utworzone, a nie w momencie pisania programu. Powyższe zadania realizuje właśnie wzorzec fabryki. Implementacja fabryki w C++ polega na napisaniu klasy, która ma metodę przyjmującą identyfikator określający rodzaj obiektu do utworzenia. Metoda tworzy na stosie i zwraca zadany obiekt.
Napisz klasę o nazwie "Fabryka", która zawiera jedną **publiczną metodę statyczną**:

```
Owoc* uprawiajOwoc(string rodzaj)
```

W zależności od podanego napisu, metoda utworzy i zwróci wskaźnik na obiekt typu Jablko, Gruszka albo Banan. W przypadku podania nieznanego identyfikatora, metoda zwroci wskaźnik pusty. Działanie fabryki pozwoli przetestować zmodyfikowana funkcja main, w której ostatni z elementów tworzony jest w trakcie wykonywania programu na podstawie identyfikatora podanego przez użytkownika.

```
int main()
{
        //wektor wskaznikow na klase abstrakcyjna?
        vector<Owoc*> v;
        // v.push_back(new Jablko);
        // v.push_back(new Gruszka);
        // v.push_back(new Banan);
        v.push_back(Fabryka::uprawiajOwoc("jablko"));
        v.push_back(Fabryka::uprawiajOwoc("gruszka"));
        v.push_back(Fabryka::uprawiajOwoc("banan"));
        cout << "Podaj nazwe..." << endl;
        string nazwa;
        cin >> nazwa;
        v.push_back(Fabryka::uprawiajOwoc(nazwa));

        for(auto owoc : v)
        {
                if(owoc)
                        owoc->powitanie();
        }
        //dostep do metody w klasie abstrakcyjnej
        v[0]->Owoc::powitanie();
        //zawsze pamietamy o zwolnieniu pamieci
        for(int ii=v.size(); ii>0; ii--)
        {
                if(v[ii])
                        delete v[ii];
        }

        return 0;
}
```