

Пусть требуется считать параметры калибровочной функции из файла и вычислить значения функции в разных точках. Определены два типа функций:

1. экспоненциальная

$$y = a + b \exp(c(x - x_0)),$$

2. полиномиальная

$$y = \sum_{n=0}^N c_n x^n.$$

Для калибровочной функции определён интерфейс:

```
class CalibFunction
{
public:
    virtual ~CalibFunction() = default;
    virtual double evaluate(double x) const noexcept = 0;
    virtual void load(std::istream &str) = 0;
};
```

Метод `evaluate` вычисляет значение функции в заданной точке. Метод `load` загружает параметры функции из потока (конфигурационного файла). Если в процессе загрузки возникла ошибка, то нужно выбросить исключение типа `std::runtime_error`.

Напишите реализации интерфейса `Exponential` и `Polynomial` для экспоненциальной и полиномиальной функции соответственно. Также напишите функцию-фабрику:

```
std::unique_ptr<CalibFunction> factory(std::istream &str);
```

Эта функция должна в зависимости от указанного в конфигурации параметра создать объект нужного типа (`Exponential` или `Polynomial`) и вернуть его в виде указателя на интерфейс.

Создание функций-фабрик или классов-фабрик является достаточно популярным приёмом в программировании. Более подробно можно почитать здесь: <https://refactoring.guru/ru/design-patterns/factory-method>.

Пример функции `main`, которая табулирует значения функции:

```
int main()
{
    std::fstream file;
    file.open("config.txt");
    if(!file.is_open())
        throw std::runtime_error("Cannot open file");
    std::unique_ptr<CalibFunction> func = factory(file);
    for(double x = 0; x <= 10; x += 0.25)
        std::cout << "f(" << x << ") = " << func->evaluate(x) << '\n';
    return 0;
}
```

Структура конфигурационного файла для экспоненциальной функции:

```
exp a b c x0
```

Для полиномиальной:

```
poly N c0 c1 c2 ... cN
```

Первое поле (`exp` или `poly`) представляет собой строку, которая указывает тип функции.

Остальные поля – числовые и выражают параметры функции.