Konstruktor klasy MoedlARX sprawdza poprawność wprowadzonych danych oraz wpisuje je do obiektu:

```
ModelARX::ModelARX(int rzA, int rzB,vector<double> mA, vector<double> mB, int op) {
    dA = rzA;
    dB = rzB;

    //cout << mA.size() << " i " << rzA << endl;
    if (mA.size() == rzA) {
        A = mA;
    }
    else {
        cout << "rząd A niezgodny z macierzą A" << endl;
        return;
    }

    //inicjalizacja macierzy B jeśli nie została ona podana
    if (mB.size() == rzB) {
        B = mB;
    }

    //inicjalizacja macierzy pamięci wyjść i wejść
    for (int i = 0; i < rzB + op; i++) {
        u.push_back(0);
    }
    for (int i = 0; i < rzA; i++) {
        y.push_back(0);
    }
    //cout << "rzB" << rzB << endl;
    //cout << "sizeof u= " << u.size() << endl <<u[0];

    d = op;


}
```

Wektory A i B są przechowywane w formie vector<double> w celu łatwego wprowadzania dowolnej wielkości.

```
class ModelARX : public siso {
private:

    int dA = 1;
    int dB = 1;
    vector<double> A = { 1 };
    vector<double> B = { 1 };

    vector<double> u;
    vector<double> y;

    int d = 0;
public:
    double Symuluj(double = 0, double = 0);
    ModelARX(int sizeA, int sizeB, vector<double>, vector<double>, int op);

};
```

Wartości poprzednich wyjść i wejść także zapisywane są w postaci vectorów.

Funkcja symulacji modelu ARX wygląda następująco:

```
}
double ModelARX::Symuluj(double wartosc, double zaklocenie) {
    double tempM = 0;
    double tempL = 0;

    for (int i = (u.size()) - 1 - 1 + d; i >= 0; i--) {
        u[i + 1] = u[i];
        //cout <<"u[1]"<< u[1] << endl;
    }
    u[0] = wartosc;
    //cout << "u[0]" << u[0] << endl;
    for (int i = 0; i < B.size(); i++)
    {
        tempL = tempL + (B[i] * u[i + d]);
        //cout << "tempL dla i=" << i <<" wartosc: " << tempL << endl;
    }
    for (int i = 0; i < A.size() - 1; i++)
    {
        tempM = tempM - (A[i+1] * y[i]);
    }
    for (int i = y.size() - 1 - 1; i >= 0; i--) {
        y[i + 1] = y[i];
    }
    y[0] = tempL + tempM + zaklocenie;
    cout << "Sygnał wyjściowy obiektu: " << y[0] << endl;
    return y[0];
}
```

Wykorzystana została postać różnicowa równania obiektu.

Konstruktor klasy Regulator wpisuje wartości regulatora do obiektu

```
}
Regulator::Regulator(double kP, double kI, double kD) {
    P = kP;
    I = kI;
    D = kD;
}
```

Symulacja regulatora wygląda w następujący sposób:

```
double Regulator::Symuluj(double wejscie, double zaklocenie) {
    u[1] = u[0];
    u[0] = wejscie;
    //cout << "wejscie regulator"<< u[0] << endl;
    double dU = u[0] - u[1];
    double wyjscie = P * u[0] + I * (u[0] * u[1]) / 2 + D * dU;
    //   wyjscie = wejscie * PID
    cout << "sygnał wyjściowy regulatora " << wyjscie << endl;
    return wyjscie;
}
```

Całość symulacji została przetestowana w następujący sposób:

```
    //Ręczna definicja modelu oraz regulatora
    vector<double> A = { 2 };
    vector<double> B = { 5 , 2 };
    ModelARX newARX(1, 2, A, B, 0);
    Regulator newRegulator(1, 1, 1);
    double wyjscie = newARX.Symuluj(10, 0);
    double wyjscieReg = newRegulator.Symuluj(wyjscie, 0);
    double wyjscie2 = newARX.Symuluj(wyjscieReg, 0);
    cout <<"koniec programu: "<< wyjscie2 << endl;


    return 0;
```

Efekt końcowy:

```
Microsoft Visual Studio Debug Console
Sygna? wyj?ciowy obiektu: 50
sygna? wyj?ciowy regulatora 100
Sygna? wyj?ciowy obiektu: 520
koniec programu: 520

D:\Uczelnia\PSS\PSS1\x64\Debug\PSS1.exe (process 3128) exited with code 0.
Press any key to close this window . . .
```