1 B.

1.1

```
//
// Created by matematyk60 on 17.06.17.
//


#include <iostream>


using namespace std;

template<class T>
class Matrix {
public:
    Matrix(int rows, int cols){
        if(rows < 1 || cols < 1){
            throw "Invalid size!";
        }
        this->rows = rows;
        this->cols = cols;
        matrix = new T[rows*cols];
        for(int i = 0 ; i < rows*cols ; i++){
            matrix[i] = T(2.3333+i);
        }
    }

    ~Matrix(){
        delete [] matrix;
    }

    T& operator()(int r, int c){
        if(r < 1 || r > rows ||
        c < 0 || c > cols){
            throw "InvalidIndex";
        }
        return matrix[(r-1)*cols+c-1];
    }

    Matrix submatrix(int r1, int r2,
    int c1, int c2){
        if(c2<c1 || r2<r1 || c2>cols ||
        r2>rows || r1 < 1 || c1 < 1){
            throw "InvalidIndex";
        }
        Matrix<T> copy(r2-r1+1,c2-c1+1);
        int p = 0;
        for(int i = r1 ; i <= r2 ; i++){
            for(int j = c1 ; j <= c2 ; j++){
                copy.matrix[p] =
                this->operator()(i,j);
                p++;
            }
        }
        return copy;

    }

    void PrintMatrix(){
        for(int i = 1 ; i < rows+1 ; i++){
            for(int j = 1 ; j < cols+1 ; j++){
                std::cout <<
                this->operator()(i,j) << " ";
            }
            std::cout << std::endl;
        }
    }

private:
    int rows;
    int cols;
    T* matrix;
};
```

1.2

```
//
// Created by matematyk60 on 17.06.17.
//

#include <utility>
#include <map>
#include <vector>
#include <list>
#include <cmath>
#include <iostream>

struct Miasto{
public:
    Miasto(const char *nazwa,
    double x = 0, double y= 0){
        this->x = x;
        this->y = y;
        this->nazwa = nazwa;
    }
    const char *nazwa;
    double x;
    double y;
};

struct Droga{
    int m1;
    int m2;
    double d;
    Droga(int m1, int m2, double d){
        this->m1 = m1;
        this->m2 = m2;
        this->d = d;
    }
};

class Mapa {
public:
    Mapa(){
        this->nextId=0;
    }

    int DodajMiasto(const char *nazwa,
    double x, double y){
        int id = nextId;
        miasta.insert(std::pair<int,
        Miasto>(id,Miasto(nazwa,x,y)));
        nextId++;
        return id;
    }
    bool DodajDroge(int m1,
    int m2, double dl){
        if(miasta.find(m1)== miasta.end()
        || miasta.find(m2) == miasta.end()){
            return false;
        }
        drogi.emplace_back(Droga(m1,m2,dl));
        return true;
    }

    void sasiedzi(int m, std::list<int>&sa){
        for(auto &n : drogi){
            if(n.m1 == m){
                sa.emplace_back(n.m2);
            } else if (n.m2 == m){
                sa.emplace_back(n.m1);
            }
        }
    }

    double odleglosc(int m1, int m2){
        Miasto miasto1 = miasta.find(m1)->second;
        Miasto miasto2 = miasta.find(m2)->second;
        return std::sqrt((miasto2.x - miasto1.x)
        *
        (miasto2.x - miasto1.x)+(miasto2.y - miasto1.y)
        *
        (miasto2.y - miasto1.y));
    }


private:
    std::map<int, Miasto> miasta;
    std::vector<Droga> drogi;
    int nextId;

};
```

1.3

```
//
// Created by matematyk60 on 17.06.17.
//

#include <iostream>
#include <list>
#include <tgmath.h>
#include <vector>


class Wielomian {
public:
    Wielomian(std::initializer_list<double> wielo){
        for(auto n : wielo){
            wielomian.emplace_back(n);
        }
    }

    Wielomian(std::list<double> wielo){
        for(auto n : wielo){
            wielomian.emplace_back(n);
        }
    }
    double wartosc(double x){
        double answer = 0;
        for(int i = 0 ; i < wielomian.size() ; i++){
            answer += wielomian[i]*pow(x,i);
        }
        return answer;
    }

    void ustawWspolczynnik(int n, double value){
        wielomian[n] = value;
    }

    friend std::ostream& operator<<(std::ostream&
    output, Wielomian& w1){
        for(int i = 0 ; i < w1.wielomian.size()
        ; i++){
            output << w1.wielomian.at(i) << "x^" << i ;
            if(i != w1.wielomian.size()-1){
                output << " + ";
            }
        }
        output << std::endl;
        return output;
    }

    friend std::istream& operator>>
    (std::istream &input, Wielomian& w1){
        std::list<double> lista;
        while(input.peek() != -1){
            lista.emplace_back(
            Wielomian::Wspolczynnik(input));
            Wielomian::SkipPlus(input);
        }
        w1 = Wielomian(lista);
        return input;
    }

    static double Wspolczynnik(
    std::istream& input){
        std::stringstream ss;
        std::string pa = "";
        double a = 0;
        while(input.peek() != 'x'){
            pa += input.get();
        }
        ss << pa;
        ss >> a;
        while(input.peek() != ' '
        && input.peek() != -1){
            input.ignore();
        }
        return a;
    }

    static void SkipPlus(std::istream& input){
        while(input.peek() == ' ' ||
        input.peek() == '+'){
            input.ignore();
        }
    }

    Wielomian operator+(Wielomian w2){
        std::list<double> lista;
        double a1, a2;
        std::vector<double>::iterator it =
        this->wielomian.begin();
        std::vector<double>::iterator it2 =
        w2.wielomian.begin();
        while(it != this->wielomian.end() ||
        it2 != w2.wielomian.end()){
            if(it != this->wielomian.end()){
                a1 = *it;
            } else{
                a1 = 0;
            }
            if(it2 != w2.wielomian.end()){
                a2 = *it2;
            } else{
                a2 = 0;
            }
            lista.emplace_back(a1+a2);
            if(it != this->wielomian.end()) {
                it++;
            }
            if(it2 != w2.wielomian.end()) {
                it2++;
            }
        }
        return Wielomian(lista);
    }

    Wielomian operator*(Wielomian w2){
        std::vector<Wielomian> wielomiany;
```

```
std::list<double> lista;
for(int i = 0 ; i <
this->wielomian.size() ; i++){
    lista.clear();
    for(int j = 0 ; j < i ; j++){
        lista.emplace_back(0);
    }
    for(int j = 0 ; j <
    w2.wielomian.size() ; j++){
        lista.emplace_back(
        this->wielomian[i]*w2.wielomian[j]);
    }
    wielomiany.emplace_back(Wielomian(lista));
}
Wielomian answer = *(wielomiany.begin());
for(auto it = wielomiany.begin()+1 ;
it != wielomiany.end() ; it++ ){
    answer = answer+*it;
}
return answer;
}


  private:
      std::vector<double> wielomian;

  };
```

1.4

```
#include <cstdio>

class A{
public:
    virtual void f(){
        printf("~A.f \n");
    }
    virtual ~A(){f();}
};

class B : public A {
public:
    void f(){
        printf("~B.f\n");
    }
    ~B(){f();
    printf("sss\n");}
};

B b;

int main() {
    A*a= new B();
    printf("M \n");
    delete a;
    return 0;
}
/*M
~B.f
sss
~A.f
~B.f
sss
~A.f */
```