1 C.

1.1

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

#include <iostream>

class Animal{
public:
    virtual ~Animal(){};
    virtual void Print() const = 0;
};

class Cat : public Animal{
public:
    ~Cat(){}

    void Print() const override {
        std::cout<<"kot" << std::endl;
    }
};

class Frog : public Animal{
public:
    ~Frog(){}

    void Print() const override {
        std::cout<<"zaba" << std::endl;
    }
};

class Dog : public Animal{
public:
    ~Dog(){}

    void Print() const override {
        std::cout<<"pies" << std::endl;
    }
};

class AnimalList {
    struct Node{
        Animal *value;
        Node* next;
    };
public:
    AnimalList(){
        list = nullptr;
    }
    AnimalList(const AnimalList& al1){
        //brakuje konstruktora kopiujacego
    }

    ~AnimalList(){
        if(list == nullptr){
            return;
        }
        Node* actual = list;
        Node* next = actual->next;
        while(next != nullptr){
            delete actual->value;
            delete actual;
            actual = next;
            next = actual->next;
        }
        delete actual->value;
        delete actual;
        list = nullptr;
    }
    void add(Animal *a){
        if(list == nullptr){
            list = new Node;
            list->value = a;
            list->next = nullptr;
            return;
        }
        Node* n = list;
        while(n->next != nullptr){
            n = n->next;
        }
        n->next = new Node;
        n->next->value = a;
        n->next->next = nullptr;
    }

    void Print(){
        Node* n = list;
        while(n->next != nullptr){
            n->value->Print();
            n = n->next;
        }
        n->value->Print();
    }



private:
    Node *list;
};
```

1.2

```
//
// Created by matematyk60 on 17.06.17.
//
#include <list>
#include <map>
#include <iostream>

using namespace std;

class Slownik{
public:
    void dodaj(const char* term,
    const char* znaczenie) {
        if (slownik.find(term) !=
        slownik.end()) {
            slownik.find(term)->
            second.emplace_back(znaczenie);
        } else {
            slownik.insert(std::pair<string,
            std::list<string>>(term,
            std::list<string>{znaczenie}));
        };
    }

    list<string> szukaj(const char *term){
        std::map<string,std::list<string>>
        ::iterator it = slownik.find(term);
        if(it != slownik.end()){
            return it->second;
        } else{
            return list<string>();
        }
    }

    void usun(const char *term){
        slownik.erase(term);
    }

    void usun(const char *term,
    const char *znaczenie){
        std::map<string,std::list<string>>
        ::iterator it = slownik.find(term);
        for( auto it2 = it->second.begin();
        it2 != it->second.end() ; it2++){
            if(*it2 == znaczenie){
                it->second.erase(it2);
                break;
            }
        }
        if(it->second.empty()){
            slownik.erase(term);
        }

    }
private:
    std::map<string,std::list<string>> slownik;
};
```

1.3

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

#include <vector>
#include <iostream>

class Mat : public std::vector<double*>{
public:
    Mat(){}

    double *addRow(int size){
        this->emplace_back(new double[size]);
        sizes.emplace_back(size);
        double *p = this->at(this->size()-1);
        for(int i = 0 ; i < sizes.at(this->size()-1)
        ; i++){
            *(p+i) = 1;
        }
        return this->at(this->size()-1);
    }

    void deleteRow(int r) {
        if (r > this->size() || r < 1) {
            throw "InvalidIndex";
        }
        delete[] this->at(r - 1);
        this->erase(this->begin() + r - 1);
        sizes.erase(sizes.begin() +r-1);
    }

    double &operator()(int row, int col){
        double * p = this->at(row-1);
        if(col > sizes.at(row-1)){
            throw "InvalidIndex";
        }
        return *(p+col-1);
    }

    void Print(){
        for(int i = 0 ; i < this->size() ;
        i++){
            for(int j = 0 ; j < sizes[i] ;
            j++){
                std::cout<<*(this->at(i)+j)
                << " ";
            }
            std::cout << "\n";
        }
    }

    Mat operator +(Mat& m2){
        Mat answer;
        long lim1 = this->size();
        long lim2 = m2.size();
        int i1, i2;
        for(int i = 0; i <
        std::max(lim1,lim2) ; i++){
            if(i >= lim1){
                i1 = 0;
            } else{
                i1 = this->sizes[i];
            }
            if(i >= lim2){
                i2 = 0;
            } else{
                i2 = m2.sizes[i];
            }
            answer.addRow(std::max(i1, i2));
        }
        double v1, v2;
        for(int i = 0 ; i < answer.size() ; i++){
            lim1 = this->sizes[i];
            lim2 = m2.sizes[i];
            for(int j = 0 ; j < std::max(
            lim1,lim2) ; j++){
                if(j >= lim1){
                    v1 = 0;
                } else{
                    v1 = this->
                    operator()(i+1,j+1);
                }
                if(j >= lim2){
                    v2 = 0;
                }else {
                    v2 = m2(i+1,j+1);
                }
                answer(i+1,j+1) = v1+v2;
            }
        }
        return answer;

    }

private:
    std::vector<int> sizes;
};
```

1.4

```
#include <cstdio>

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

class B : public A
{
public:
    A*a;
    void f(){
        printf("B.f \n");
    }
    B(){
        f();
        a = new A();
        throw 0;
    }
};



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