1 A.

1.1

```
//
// Created by matematyk60 on 18.06.17.
//

#include <iostream>

class LE{
public:
    LE(int rozm){
        tablica = new int[rozm];
        rozmiar= rozm;
        licznik = 0;
        next = nullptr;
    }
    ~LE(){
        if(next != nullptr){
            delete next;
        }
        delete [] tablica;
    }

    void addElement(int value){
        if(licznik >= rozmiar){
            next->addElement(value);
        } else{
            tablica[licznik] = value;
            licznik++;
            if(licznik == rozmiar){
                next = new LE(rozmiar);
            }
        }
    }

    int pobierz(int n){
        if(n>rozmiar){
            return next->pobierz(n-rozmiar);
        } else{
            return tablica[n-1];
        }
    }
private:
    int*tablica;
    //tablica na dane
    int rozmiar;
    //rozmiar tablicy
    int licznik;
    //ile zajetych
    LE *next;
};

class ListaTablic{
public:
    ListaTablic(){
        lista = new LE(5);
    }
    ~ListaTablic(){
        delete lista;
    }
    void Dodaj(int value){
        lista->addElement(value);
    }

    int pobierz(int n){
        return lista->pobierz(n);
    }
private:
    LE *lista;
};

int main(){
    ListaTablic l1;
    l1.Dodaj(2);
    std::cout << l1.pobierz(8);
}
```

1.2

```
//
// Created by matematyk60 on 18.06.17.
//
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

using ::std::string;

class Miasto{
public:
    friend class ZbiorMiast;
    Miasto(string nazwa, double x, double y){
        this->nazwa = nazwa;
        this->x = x;
        this->y = y;
    }
    string getNazwa()const{
        return nazwa;
    }
    /*double getX()const {return x;}
    double getY()const {return y;}*/

private:
    std::string nazwa;
    double x;
    double y;
};

class ZbiorMiast{
public:
    bool dodaj(const Miasto& m){
        string name = m.nazwa;
        std::vector<Miasto*>::iterator it =
        std::find_if(miasta.begin(), miasta.end(),
        [name](Miasto* tmp)->bool{
        return tmp->nazwa == name; });
        if(it == miasta.end()){
            miasta.emplace_back(new Miasto(m));
            return true;
        } else{
            (*it)->x = m.x;
            (*it)->y = m.y;
            return false;
        }
    }
    Miasto *znajdz(const char *name){
        auto it = std::find_if(miasta.begin(),
        miasta.end(), [name](Miasto*tmp)->bool{
        return tmp->nazwa == name;});
        if(it == miasta.end()){
            return nullptr;
        } else {
            return (*it);
        }
    }
    void dodaj(const ZbiorMiast&z){
        for(auto n : z.miasta){
            this->dodaj(*n);
        }
    }

    void usunSpoza(const ZbiorMiast&z){
        int i = 0;
        std::cout<<miasta.size()<< "\n\n";
        for(std::vector<Miasto*>::iterator n =
        miasta.begin() ; n != miasta.end() ; ){
            auto it = std::find_if(z.miasta.begin(),
            z.miasta.end(), [n](Miasto*tmp)->bool{
            return tmp->nazwa == (*n)->nazwa;});
            if(it == z.miasta.end()){
                delete (*n);
                miasta.erase(n);
            } else{
                n++;
            }
        }
    }

    void Print(){
        for(auto n : miasta){
            std::cout << n->nazwa << " " <<
            n->x << " " << n->y << " | ";
        }
        std::cout << "\n";
    }

private:
    std::vector<Miasto*> miasta;
};
```

1.3

```
//
// Created by matematyk60 on 18.06.17.
//

#include <cstring>
#include <iostream>

class TextArray {
public:
    TextArray(int size_ = 8){
        max_size = size_;
        tab= new char*[max_size];
        size = 0;
    }

    ~TextArray(){
        for(int i = 0 ; i < size ; i++){
            delete [] *(tab+i);
        }
        delete [] tab;
    }

    TextArray(const TextArray& n){
        max_size = n.max_size;
        tab = new char*[max_size];
        size = n.size;
        for(int i  = 0 ; i < n.size ; i++){
            *(tab+i) = new
            char[std::strlen(*(n.tab+i))+1];
            std::strcpy(*(tab+i),*(n.tab+i));
        }
    }

    TextArray& operator=(const TextArray& n){
        if(this == &n){
            return *this;
        }
        for(int i = 0 ; i < size ; i++){
            delete [] *(tab+i);
        }
        delete [] tab;
        max_size = n.max_size;
        size = n.size;
        tab = new char*[max_size];
        for(int i  = 0 ; i < n.size ; i++){
            *(tab+i) =
            new char[std::strlen(*(n.tab+i))+1];
            std::strcpy(*(tab+i),*(n.tab+i));
        }
        return *this;

    }

    void resize(){
        char** new_ = new char*[max_size*2];
        for(int i = 0 ; i < size ; i++){
            *(new_+i) = new
            char[std::strlen(*(tab+i))+1];
            std::strcpy(*(new_+i),*(tab+i));
        }
        for(int i = 0 ; i < size ; i++){
            delete [] *(tab+i);
        }
        delete [] tab;
        tab = new_;
        max_size = max_size*2;
    }

    bool add(const char*napis) {
        if(size == max_size){
            resize();
        }
        *(tab+size) = new char[std::strlen(napis)+1];
        std::strcpy(*(tab+size),napis);
        size+=1;
        return true;
    }

    const char* get(int n){
        if(n < 0 || n > size){
            throw "InvalidIndex";
        }
        return tab[n-1];
    }

    void Print(){
        for(int i = 0 ; i < size; i++){
            std::printf(*(tab+i));
            std::cout << "\n";
        }
    }
private:
    char** tab;
    int size;
    int max_size;
};
```

1.4

```
//
// Created by matematyk60 on 18.06.17.
//

#include <cstdio>

class A {
    int i;
public:
    A(int _i = 0) : i(_i){
        printf("A%d\n",i);
    }
    ~A(){
        printf("~A%d\n",i);
    }
};

class B : public A{
    int x;
    A a;
public:
    B(int _x ):A(1),x(_x){};
    ~B(){
        printf("~B %d\n",x);
    }
};

B b(5);

int main(){
    A*ptr = new B(3);
    delete ptr;
    return 0;
}

/*A1
A0
A1
A0
~A1
~B 5
~A0
~A1*/
```