МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Кафедра “Обчислювальна техніка та програмування”

Розрахункове завдання з дисципліни
«Програмування»

Пояснювальна записка

Виконав:
студент групи КІТ-120Д
Стеценко М.О.

Перевірив:
Пасько Д.А.

Харків 2021

Тема: Розробка інформаційно-довідкової системи

Мета: закріпити отримані знання з дисципліни «Програмування» шляхом виконання типового комплексного завдання.

1. Вимоги

1.1. Розробник

- Стеценко Микита Олександрович;
- Студент групи КІТ-120Д;
- 6-червня-2021;

1.2. Загальне завдання:

1. З розділу «Розрахункове завдання/Індивідуальні завдання», відповідно до варіанта завдання, обрати прикладну галузь
2. Для прикладної галузі розробити розгалужену ієрархію класів, що описана у завдання та складається з одного базового класу та двох спадкоємців. Класи повинні мати перевантажені оператори введення-виведення даних та порівняння
3. Розробити клас-список *List*, що буде включати до себе масив (STL-колекцію) вказівників до базового класу. А також базові методи роботи зі списком: очистка, відображення/додання/видалення/отримання/ оновлення елементу
4. Розробити клас-контролер *Controller*, що буде включати колекцію розроблених класів та наступні методи роботи з колекцією: читання даних з файлу, запис даних у файл, сортування елементів у контейнері за заданими полем та напрямом, пошук елементів за критеріями, вказаними у індивідуальному завданні
5. Розробити клас *Menu*, який має відображати діалогове меню для демонстрації реалізованих функцій класу контролера

Додаткові вимоги:

- Виконати перевірку вхідних даних за допомогою регулярних виразів
- Критерій для пошуку та сортування задавати у вигляді функтора
- Розробити клас-тестер *ControllerTest*, основною метою якого буде перевірка коректності роботи контролера

2. Опис програми

2.1. Функціональне призначення

Програма призначена для роботи з інформаційно-довідковою системою.

Результат зберігається у змінній menu.

Демонстрація знайдених результатів передбачає виконання програми та виведення результатів до консолі.

2.2. Важливі фрагменти

```
//Код класу "Лампочка"

#ifndef BULB
#define BULB
#include <string>
#include <sstream>
using std::string;
using std::endl;

class Bulb {
protected:

    string status;
    int state;
    string manufacturer;
    int death;
    int watt;
    int color;
    string shape;
    string plinth;
    static const string shapes[5];
    static const string plinths[3];
public:
    Bulb();
    Bulb(const Bulb& copy);

    Bulb(string status, int state, string man, int death, int watt, int col,
int sh, int pl);
    virtual ~Bulb();

    void set_status(const string status);
    string get_status() const;

    void set_state(const int state);
    int get_state () const;

    void set_manufacturer(const string man);
    string get_manufacturer() const;

    void set_death (const int death);
    int get_death () const;

    void set_watt (const int watt);
    int get_watt () const;

    void set_color (const int col);
    int get_color() const;

    void set_shape(const int sh);
    string get_shape() const;

    void set_plinth(const int pl);
    string get_plinth() const;

    virtual string toString() const;
    virtual void fromString(string l);
    virtual Bulb& Copy() const = 0;
    virtual Bulb& operator= (const Bulb &other);
    virtual bool operator< (const Bulb &other);
    friend std::ostream& operator<< (std::ostream &out, const Bulb &bulb);
    friend std::istream& operator>> (std::istream &in, Bulb &bulb);
    static Bulb* BulbById(int id);
};
```

```
#endif

// Код спадкоємця "Вічна лампочка"

#ifndef ETERNAL
#define ETERNAL
#include "bulb.h"
using std::string;
using std::endl;
class Eternal: public Bulb {
protected:
    string eternal;
public:
    Eternal();
    Eternal(const Eternal& copy);
    Eternal(string status, int state, string man, int death, int watt, int
col, int sh, int pl, string et);
    ~Eternal();

    void set_eternal(const string et);
    string get_eternal() const;

    Eternal& operator= (const Eternal &other);
    Eternal& Copy() const override final;
    void fromString(string l) override final;
    string toString() const override final;
};
#endif

// Код спадкоємця "Розумна лампочка"

#ifndef SMART
#define SMART
#include "bulb.h"
using std::string;
using std::endl;
class Smart : public Bulb {
private:
    static const string type[2];
protected:
    string smart;
    string wireless;
    string microcontroller;
    string hex;
public:
    Smart();
    Smart(const Smart& copy);

    Smart(string status, int state, string man, int death, int watt, int col,
int sh, int pl, string sm, string wl, int mc, string hx);
    ~Smart();
    void set_smart(const string sm);
    string get_smart() const;
    void set_wireless(const string wl);
    string get_wireless() const;
    void set_microcontroller(const int mc);
    string get_microcontroller() const;
    void set_hex(const string hx);
    string get_hex() const;

    string toString() const override final;
    void fromString(string l) override final;
    Smart& operator= (const Smart &other);
```

```
    Smart &Copy() const override final ;
};
#endif
```

Bulb.cpp

```
// Код реалізації класу "Лампочка"

#include "bulb.h"
#include "smart.h"
#include "eternal.h"
const string Bulb::shapes[] = {"Candle", "Tubular", "Globe", "Pear",
"Ogive"};
const string Bulb::plinths[] = {"E14", "E27", "E40"};
Bulb::Bulb() : state(0), death(0), watt(0), color(0), shape(shapes[0]),
plinth(plinths[0]) {
}
Bulb::Bulb(const Bulb& copy)
    : status(copy.status),
      state(copy.state),
      manufacturer(copy.manufacturer),
      death(copy.death),
      watt(copy.watt),
      color(copy.color),
      shape(copy.shape),
      plinth(copy.plinth) {
}
Bulb::Bulb(string sts, int ste, string man, int dth, int w, int col, int sh,
int pl)
    : status(sts),
      state(ste),
      manufacturer(man),
      death(dth),
      watt(w),
      color(col),
      shape(shapes[sh]),
      plinth(plinths[pl]) {
}
Bulb::~Bulb() {
}
void Bulb::set_status(const string sts) {
    status = sts;
}
string Bulb::get_status() const {
    return status;
}
void Bulb::set_state(const int ste) {
    state = ste;
}
int Bulb::get_state() const {
    return state;
}
void Bulb::set_manufacturer(const string man) {
    manufacturer = man;
}
string Bulb::get_manufacturer() const {
    return manufacturer;
}
void Bulb::set_death(const int dth) {
    death = dth;
}
int Bulb::get_death() const {
```

```
    return death;
}
void Bulb::set_watt(const int w) {
    watt = w;
}
int Bulb::get_watt() const {
    return watt;
}
void Bulb::set_color(const int col) {
    color = col;
}
int Bulb::get_color() const {
    return color;
}
void Bulb::set_shape(const int sh) {
    shape = shapes[sh];
}
string Bulb::get_shape() const {
    return shape;
}
void Bulb::set_plinth(const int pl) {
    plinth = plinths[pl];
}
string Bulb::get_plinth() const {
    return plinth;
}
string Bulb::toString() const
{
    std::stringstream l;
    l << "\n\tStatus: " << status << ";\n\tState: " << state <<
";\n\tManufacturer: " << manufacturer << ";\n\tNumber of starts before
burnout: " << death << ";\n\tConsumption in watts: " << watt << ";\n\tLight
temperature: " << color << ";\n\tShape: " << shape << ";\n \tType of plinth:
" << plinth << "." << endl;
    return l.str();
}
void Bulb::fromString(string l)
{
    int sh, pl;
    std::stringstream str;
    str << l;
    str >> status >> state >> manufacturer >> death >> watt >> color >> sh >>
pl;
    set_shape(sh);
    set_plinth(pl);
}
Bulb& Bulb::operator= (const Bulb &other)
{
    if (this == &other)
    {
        return *this;
    }
    status = other.status;
    state = other.state;
    manufacturer = other.manufacturer;
    death = other.death;
    watt = other.watt;
    color = other.color;
    shape = other.shape;
    plinth = other.plinth;
    return *this;
}
bool Bulb::operator< (const Bulb &other) {
```

```
    return (this->state < other.state);
}
std::ostream& operator<< (std::ostream &out, const Bulb &bulb) {
    out << bulb.toString();
    return out;
}
std::istream& operator>> (std::istream &in, Bulb &bulb) {
    string l;
    getline(in, l);
    bulb.fromString(l);
    return in;
}
Bulb* Bulb::BulbById(int id) {
    switch (id) {
    case 0:
            return new Eternal();
    case 1:
        return new Smart();
    default:
        return nullptr;
    }
}
```

---

bulbar.cpp

```
// Код реалізації класу "Колекція лампочок"

#include "bulbarr.h"
BulbArr::BulbArr() {

}
BulbArr::~BulbArr() {
    for(vector<Bulb*>::iterator it = bulbs.begin(); it != bulbs.end(); it++)
        delete *it;

    bulbs.clear();
}
Bulb& BulbArr::getBulb(size_t index) const {
    return *bulbs[index];
}
size_t BulbArr::getSize() const {
    return bulbs.size();
}
Bulb& BulbArr::operator[] (const int index) const {
    return *bulbs[index];
}
void BulbArr::showAll() const {
    int i = 1;
    for (Bulb* lightbulb : bulbs)
        cout << "Bulb " << i++ << ": " << endl << lightbulb->toString() <<
endl;
}
void BulbArr::addBulb(Bulb& bulb)
{
    Bulb *lightbulb = &bulb.Copy();
    bulbs.push_back(lightbulb);
}
void BulbArr::removeBulb(int index) {
    delete bulbs[index];
    bulbs.erase(bulbs.begin() + index);
}
void BulbArr::Merger(BulbArr& lightbulb) {
    size_t size = lightbulb.bulbs.size();
```

```
    for (size_t i = 0; i < size; ++i)
        addBulb(**(lightbulb.bulbs.begin() + i));
}
void BulbArr::readFromFile(string fileName) {
    std::ifstream in(fileName);
    if(in.is_open()) {
        cout << "Reading the file ... " << endl << fileName << endl;
        in >> *this;
        in.close();
    }
    else {
        cout << "ERROR... We cannot find the file... " << endl << fileName <<
endl;
    }
}
void BulbArr::writeToFile (string fileName) {
    std::ofstream out;
    out.open(fileName);
     if(out.is_open()) {
        cout << "Writing to the file ... " << endl << fileName << endl;
        out << *this;
        out.close();
    }
    else {
        cout << "ERROR... We cannot find the file... " << endl << fileName <<
endl;
    }
}
std::ostream& operator<< (std::ostream &out, const BulbArr &lightbulb) {
    for (size_t i = 0; i < lightbulb.getSize(); i++)
        out << "Bulb " << i+1 << ": " << endl << lightbulb[i] << endl;
    return out;
}
std::istream& operator>> (std::istream &in, BulbArr &lightbulb) {
    Bulb *bulb;
    string h;
    std::stringstream str;
    int id;
    std::regex smart("\\s[A-Za-z]*\\s[A-Za-z0-9-]*\\s[0-9]*\\s[0-2]\\s[A-Za-
z]*\\s[A-Za-z]*\\s[A-Za-z]*\\s[A-Za-z]*");
    std::regex eternal("\\s[A-Za-z]*\\s[A-Za-z0-9-]*\\s[0-9]*\\s[0-2]\\s[A-
Za-z]*\\s[A-Za-z]*\\s[0-1]\\s[0-9]*");
    std::smatch m;
        while(in) {
        getline(in, h);
        str.clear();
        str << h;
        str >> id;
        bulb = bulb->BulbById(id);
        getline(str, h);
        switch (id) {
        case 0:
            if(regex_match(h, m, eternal))
            {
                bulb->fromString(h);
                lightbulb.addBulb(*bulb);
            }
            break;
        case 1:
            if(regex_match(h, m, smart))
            {
                bulb->fromString(h);
```

```
                lightbulb.addBulb(*bulb);
            }
            break;
        }
        delete bulb;
    }
    return in;
}
void BulbArr::find_deadbulb() const {
    int i = 1;
    for (Bulb* lightbulb : bulbs)
        if (lightbulb->get_state() == 0) {
            cout << "Bulb " << i << endl << *lightbulb;
            i++;
        }
}
void BulbArr::find_smartbulb() const {
    int i = 1;
    for (Bulb* lightbulb : bulbs) {
        Smart* h = dynamic_cast<Smart*>(lightbulb);
        if ( h != nullptr) {
            if ( h->get_smart() == "Yes") {
                cout << "Bulb " << i << endl << *lightbulb;
                i++;
            }
        }
    }
}
void BulbArr::find_fullwatt() const {
    int allwatt = 0;
    for ( Bulb* lightbulb : bulbs) {
        if ( lightbulb->get_state() != 0) {
            allwatt = allwatt + lightbulb->get_watt();
        }
    }
    cout << "Total energy consumption is " << allwatt << "watts." << endl;
}
```

---

smart.cpp

```
// Код реалізації спадкоємця "Розумна лампочка"

#include "smart.h"
const string Smart::type[] = {"ESP8266", "STM32F103"};
Smart::Smart()
    : Bulb(),
      microcontroller(type[0]) {
}
Smart::Smart(const Smart& copy)
    : Bulb(copy),
      smart(copy.smart),
      wireless(copy.wireless),
      microcontroller(copy.microcontroller),
      hex(copy.hex) {
}
Smart::Smart(string status, int state, string man, int death, int watt, int
col, int sh, int pl, string sm, string wl, int mc, string hx)
    : Bulb(status, state, man, death, watt, col, sh, pl),
      smart(sm),
      wireless(wl),
      microcontroller(type[mc]),
      hex(hx)  {
}
```

```
Smart::~Smart() {

}
void Smart::set_smart(const string sm) {
        smart = sm;
}
string Smart::get_smart() const {
        return smart;
}
void Smart::set_wireless(const string wl) {
    wireless = wl;
}
string Smart::get_wireless() const {
    return wireless;
}
void Smart::set_microcontroller(const int mc) {
    microcontroller = type[mc];
}
string Smart::get_microcontroller() const {
    return microcontroller;
}
void Smart::set_hex (const string hx) {
    hex = hx;
}
string Smart::get_hex() const {
    return hex;
}
Smart& Smart::operator= (const Smart &other)
{
    if (this == &other)
    {
        return *this;
    }
    status = other.status;
    state = other.state;
    manufacturer = other.manufacturer;
    death = other.death;
    watt = other.watt;
    color = other.color;
    shape = other.shape;
    plinth = other.plinth;
    smart = other.smart;
    wireless = other.wireless;
    microcontroller = other.microcontroller;
    hex = other.hex;
    return *this;
}
string Smart::toString() const {
        std::stringstream l;
        l << "\n\tStatus: " << status << ";\n\tState: " << state <<
";\n\tManufacturer: " << manufacturer << ";\n\tNumber of starts before
burnout: " << death << ";\n\tConsumption in watts: " << watt << ";\n\tLight
temperature: " << color << ";\n\tShape: " << shape << ";\n\tType of plinth: "
<< plinth << ";\n\tSmart: " << smart << ";\n\tWireless:" << wireless <<
";\n\tMicrocontroller:" << microcontroller << ";\n\tThe color in HEX:" << hex
<< "." << endl;
        return l.str();
}
void Smart::fromString(string l)
{
    int sh, pl, mc;
    std::stringstream str;
```

```
    str << l;
    str >> status >> state >> manufacturer >> death >> watt >> color >> sh >>
pl >> smart >> wireless >> mc >> hex;
    set_shape(sh);
    set_plinth(pl);
    set_microcontroller(mc);

}
Smart& Smart::Copy() const  {
        Smart *bulb = new Smart();
        *bulb = *this;
        return *bulb;
}
```

```
// eternal.cpp

// Код реалізації спадкоємця "Вічна лампочка"

#include "eternal.h"
Eternal::Eternal()
    : Bulb() {
}
Eternal::Eternal(const Eternal& copy)
    : Bulb(copy),
      eternal(copy.eternal) {
}
Eternal::Eternal(string status, int state, string man, int death, int watt,
int col, int sh, int pl, string et)
    : Bulb(status, state, man, death, watt, col, sh, pl),
     eternal(et) {
}
Eternal::~Eternal() {

}
void Eternal::set_eternal(const string et) {
        eternal = et;
}
string Eternal::get_eternal() const {
        return eternal;
}
Eternal& Eternal::operator= (const Eternal &other)
{
    if (this == &other)
    {
        return *this;
    }
    status = other.status;
    state = other.state;
    manufacturer = other.manufacturer;
    death = other.death;
    watt = other.watt;
    color = other.color;
    shape = other.shape;
    plinth = other.plinth;
    eternal = other.eternal;
    return *this;
}
string Eternal::toString() const {
        std::stringstream l;
        l << "\n\tStatus: " << status << ";\n\tEternal: " << eternal <<
";\n\tState: " << state << ";\n\tManufacturer: " << manufacturer <<
";\n\tNumber of starts before burnout: " << death << ";\n\tConsumption in
```

```
watts: " << watt << ";\n\tLight temperature: " << color << ";\n\tShape: " <<
shape << ";\n\tType of plinth: " << plinth << "." << endl;
        return l.str();
}
void Eternal::fromString(string l)
{
int sh, pl;
        std::stringstream str;
        str << l;
        str >> status >> state >> manufacturer >> death >> watt >> color >>
sh >> pl >> eternal;
    set_shape(sh);
    set_plinth(pl);
}
Eternal& Eternal::Copy() const {
        Eternal *bulb = new Eternal();
        *bulb = *this;
        return *bulb;
}
```

Вхідні дані

Приклад вхідних даних

```
0 On 1 Philips 0 5 1800 1 2 Yes
1 Off 0 Osram 0 15 3000 2 1 Yes No 0 118054
1 Off 1 Gauss 24 10 4000 3 0 Yes Yes 1 AC125E
1 On 1 Feron 84 20 4600 4 1 Yes 1 118038
0 Off 0 Jazzway 0 25 0 5 2 Yes
```

Використання

Результати виконання кожної задачі будуть виведені у консоль. На Рис. 1-4 зображені результати роботи програми.

```
Actions:
        1. Show bulbs;
        2. Remove bulb;
        3. Show the dead bulb;
        4. Find smart bulbs;
        5. Find total energy consumprion;
        6. Sort bulbs by watts;
        7. Merge list;
        8. Exit.
5
The total energy consumprion in watts is:
40  watts.
```

Рисунок 1 – розрахунок загального споживання енергії;

```
Actions:
        1. Show bulbs;
        2. Remove bulb;
        3. Show the dead bulb;
        4. Find smart bulbs;
        5. Find total energy consumprion;
        6. Sort bulbs by watts;
        7. Merge list;
        8. Exit.
1
Bulb 1:
        Status:         On;
        Eternal:        1;
        State:  Yes;
        Manufacturer:   Philips;
        Number of starts before burnout:        0;
        Consumption in watts:   5;
        Light temperature:      1800;
        Shape:  Candle;
        Type of plinth:         E14.
Bulb 2:
        Status:         Off;
        State:  0;
        Manufacturer:   Osram;
        Number of starts before burnout:        0;
        Consumption in watts:   15;
        Light temperature:      3000;
        Shape:  Tubular;
        Type of plinth:         E27;
        Smart:  Yes;
        Wireless:       No;
        Microcontroller:        STM32F103;
        The color in HEX:       #118064.
```

Рисунок 2 – відображення списку лампочок;

```
Actions:
        1. Show bulbs;
        2. Remove bulb;
        3. Show the dead bulb;
        4. Find smart bulbs;
        5. Find total energy consumprion;
        6. Sort bulbs by watts;
        7. Merge list;
        8. Exit.
3
Dead bulb(-s):

        Status:         Off;
        State:  0;
        Manufacturer:   Osram;
        Number of starts before burnout:        0;
        Consumption in watts:   15;
        Light temperature:      3000;
        Shape:  Tubular;
        Type of plinth:         E27;
        Smart:  Yes;
        Wireless:       No;
        Microcontroller:        STM32F103;
        The color in HEX:       #118064.
```

Рисунок 3 – пошук лампочок, котрі перегоріли;

```
        2. Remove bulb;
        3. Show the dead bulb;
        4. Find smart bulbs;
        5. Find total energy consumprion;
        6. Sort bulbs by watts;
        7. Merge list;
        8. Exit.
4
Smart bulb(-s):
Bulb:
        Status:         Off;
        State:  0;
        Manufacturer:   Osram;
        Number of starts before burnout:        0;
        Consumption in watts:   15;
        Light temperature:      3000;
        Shape:  Tubular;
        Type of plinth:         E27;
        Smart:  Yes;
        Wireless:       No;
        Microcontroller:        STM32F103;
        The color in HEX:       #118064.
Bulb:
        Status:         Off;
        State:  1;
        Manufacturer:   Gauss;
        Number of starts before burnout:        24;
        Consumption in watts:   10;
        Light temperature:      4000;
        Shape:  Globe;
        Type of plinth:         E40;
        Smart:  Yes;
        Wireless:       Yes;
        Microcontroller:        ESP8266;
        The color in HEX:       #AC125E.
```

Рисунок 4 – пошук розумних лампочок.

Висновки: У ході виконання цієї роботи було закріплено отримані знання з дисципліни «Програмування» шляхом виконання типового комплексного завдання.