

Звіт

Автор: Богданов І.Ю. КІТ-119а Дата: 17 квітня 2020

Лабораторна робота №15. РОЗУМНІ ВКАЗІВНИКИ

Тема. Розумні вказівники.

Мета: по результатах практичної роботи порівняти розумні вказівники бібліотеки STL.

1. Завдання до роботи **Індивідуальне завдання:**

Використати розумні вказівники, продемонструвати відсутність витоку пам'яті за відсутності викликів оператора delete.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: file

Клас нащадок базового класу: executable_file і non_executable_file

Клас, що повинен демонструвати композицію: options

2.2 Опис змінних

`int` n, w – службові змінні необхідні для реалізації вибору пунктів меню.

`size_t` size – поле класу file та його нащадків(розмір файлу у бітах).

`size_t` index – поле класу file та його нащадків(унікальний індекс).

`bool` is_system – поле класу options та його нащадків(чи є файл системним).

`bool` is_hidden – поле класу options та його нащадків(чи є файл прихованим).

`std::string` name – поле класу file та його нащадків(назва файлу).

`size_t` x – поле класу file та його нащадків(64 або 32 розрядна програма).

`options` `opt` поле класу `file` та його нащадків(чи є файл прихованим і системним).

`bool` `is_text` – поле класу `non_executable_file`(чи є даний файл текстовим).

`size_t` `runningtime` – поле класу `executable_file`(час виконання програми).

2.3 Опис методів

Зауваження: класи нащадки мають усі методи класу `file`.

`void` `change_ver(const int&)` – зміна значення поля `ver` змінної класу `x_version`(метод класу `x_version`).

`virtual size_t` `get_x() const` – отримання значення поля `x` змінної класу `file`(метод класу `file`).

`virtual size_t` `get_size() const` – отримання значення поля `size` змінної класу `file`(метод класу `file`).

`virtual size_t` `get_index() const` – отримання значення поля `index` змінної класу `file`(метод класу `file`).

`virtual bool` `get_sys() const` – отримання значення поля `is_system` змінної класу `file`(метод класу `file`).

`virtual bool` `get_hid() const` – отримання значення поля `is_hidden` змінної класу `file`(метод класу `file`).

`virtual std::string` `get_name() const` – отримання значення поля `name` змінної класу `file`(метод класу `file`).

`virtual void` `change_x(const size_t &x)` – зміна значення поля `x` змінної класу `file`(метод класу `file`).

`virtual void` `change_size(const size_t &sz)` – зміна значення поля `size` змінної класу `file`(метод класу `file`).

`virtual void` `change_index(const size_t &in)` – зміна значення поля `index` змінної класу `file`(метод класу `file`).

`virtual void` `change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `file`(метод класу `file`).

`virtual void` `change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `file`(метод класу `file`).

`virtual void` `change_name(const std::string&)` – зміна значення поля `name` змінної класу `file`(метод класу `file`).

`file()` – конструктор класу `file`.

`file(const file&)` – конструктор копіювання класу `file`.

`file(const int&, const int&, const int&, const bool&, const bool&, const std::string&)` – конструктор з параметрами класу `file`.

`~file()` – деструктор класу `file`.

`bool get_sys() const` – отримання значення поля `is_system` змінної класу `options` (метод класу `options`).

`bool get_hid() const` – отримання значення поля `is_hidden` змінної класу `options` (метод класу `options`).

`void change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `options` (метод класу `options`).

`void change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `options` (метод класу `options`).

`executable_file()` – конструктор класу `executable_file`.

`executable_file(const executable_file&)` – конструктор копіювання класу `executable_file`.

`executable_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const std::string&, const size_t_t&)` – конструктор з параметрами класу `executable_file`.

`~executable_file()` – деструктор класу `executable_file`.

`non_executable_file()` – конструктор класу `non_executable_file`.

`non_executable_file(const executable_file&)` – конструктор копіювання класу `non_executable_file`.

`non_executable_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const std::string&, const bool&)` – конструктор з параметрами класу `non_executable_file`.

`~non_executable_file()` – деструктор класу `non_executable_file`.

`virtual std::string get_info() const` – метод базового класу що повертає інформацію про об'єкт у вигляді строки.

`virtual size_t_t get_time() const final` – метод класу `executable_file`, повертає значення поля `runningtime`.

`virtual void change_time(const size_t_t&) final` – метод класу `executable_file`, змінює значення поля `runningtime`.

`virtual bool get_text() const final` – метод класу `non_executable_file`, повертає значення поля `is_text`.

`virtual void change_text(const bool&) final` – метод класу `non_executable_file`, змінює значення поля `is_text`.

2.4 Опис функцій

`bool operator==(const file& f1, const file& f2)` – перевантаження оператора порівняння.

`bool operator!=(const file& f1, const file& f2)` – перевантаження ще одного оператора порівняння.

`bool operator==(const executable_file& f1, const executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator!=(const executable_file& f1, const executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator==(const non_executable_file& f1, const non_executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator!=(const non_executable_file& f1, const non_executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`std::ostream& operator<<(std::ostream& os, const executable_file& f)` – перевантаження оператора виведення.

`std::ostream& operator<<(std::ostream& os, const non_executable_file& f)` – аналогічне перевантаження оператора виведення.

`std::istream& operator>>(std::istream& is, executable_file& f)` – перевантаження оператора введення.

`std::istream& operator>>(std::istream& is, non_executable_file& f)` – перевантаження оператора введення.

`std::ostream& operator<<(std::ostream& os, const file& f)` – аналогічне перевантаження оператора виведення.

3 Текст програми

Лабораторная работа 15.cpp

```
#include "file.h"
#include "Mypointer.h"
#include <memory>
#define _CRTDBG_MAP_ALLOC

void func();

int main(){
    setlocale(LC_ALL, "Russian");
    func();
    if (_CrtDumpMemoryLeaks()) {
        std::cout << "Утечка памяти обнаружена." << "\n";
    }
    else {
        std::cout << "Утечка памяти не обнаружена." << "\n";
    }
}

void func() {
    std::vector<file> vect;
    std::auto_ptr<executable_file> aptr(new executable_file);
    std::unique_ptr<file> uptr(new file);
    std::shared_ptr<non_executable_file> sptr(new non_executable_file);
    std::weak_ptr<non_executable_file> wptr = sptr;
    My_ptr<file> myptr(new file);
    vect.push_back(*aptr);
    vect.push_back(*uptr);
    vect.push_back(*sptr);
    vect.push_back(*myptr);
    std::cout << "Данные из умных указателей переписаны в вектор.\n";
    for (auto el : vect) {
        std::cout << el << "\n";
    }
}
```

file.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <regex>
#include <iomanip>
#include <vector>
#include <list>
#include <set>
#include <map>
```

```

typedef size_t size_t_t;

class options {
private:
    bool issystem;
    bool ishidden;
public:
    bool get_sys() const;
    bool get_hid() const;
    void change_sys(const bool&);
    void change_hid(const bool&);
};

class file {
protected:
    size_t_t x;
    size_t_t size;
    size_t_t index;
    options opt;
    std::string name;
public:
    int type_of_file = 0;
    virtual size_t_t get_x() const;
    virtual size_t_t get_size() const;
    virtual size_t_t get_index() const;
    virtual bool get_sys() const;
    virtual bool get_hid() const;
    virtual std::string get_name() const;
    virtual void change_x(const size_t_t&);
    virtual void change_size(const size_t_t&);
    virtual void change_index(const size_t_t&);
    virtual void change_sys(const bool&);
    virtual void change_hid(const bool&);
    virtual void change_name(const std::string&);
    virtual std::string get_info() const;
    file();
    file(const file&);
    file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const bool&, const
std::string&);
    ~file();
    file& operator=(const file& f);
    friend bool operator==(const file& f1, const file& f2);
    friend bool operator!=(const file& f1, const file& f2);
};

class executable_file final : public file {
private:
    size_t_t runningtime;
public:
    virtual size_t_t get_time() const final;
    virtual void change_time(const size_t_t&) final;
    executable_file();
    executable_file(const executable_file&);
    executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const bool&,
const std::string&, const size_t_t&);
    ~executable_file();
    virtual std::string get_info() const override;
    executable_file& operator=(const executable_file& f);
    friend bool operator==(const executable_file& f1, const executable_file& f2);
    friend bool operator!=(const executable_file& f1, const executable_file& f2);
};

class non_executable_file final : public file {
private:
    bool is_text;
public:
    virtual bool get_text() const final;
    virtual void change_text(const bool&) final;
    non_executable_file();
    non_executable_file(const non_executable_file&);
    non_executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const
bool&, const std::string&, const bool&);

```

```

~non_executable_file();
virtual std::string get_info() const override;
non_executable_file& operator=(const non_executable_file& f);
friend bool operator==(const non_executable_file& f1, const non_executable_file& f2);
friend bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
};
bool check_str(const std::string& str);
bool operator==(const file& f1, const file& f2);
bool operator!=(const file& f1, const file& f2);
bool operator==(const executable_file& f1, const executable_file& f2);
bool operator!=(const executable_file& f1, const executable_file& f2);
bool operator==(const non_executable_file& f1, const non_executable_file& f2);
bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
bool operator>(const file& f1, const file& f2);
bool operator<(const file& f1, const file& f2);
bool operator>(const executable_file& f1, const executable_file& f2);
bool operator<(const executable_file& f1, const executable_file& f2);
bool operator>(const non_executable_file& f1, const non_executable_file& f2);
bool operator<(const non_executable_file& f1, const non_executable_file& f2);
std::ostream& operator<<(std::ostream& os, const executable_file& f);
std::ostream& operator<<(std::ostream& os, const non_executable_file& f);
std::istream& operator>>(std::istream& is, executable_file& f);
std::istream& operator>>(std::istream& is, non_executable_file& f);
std::ostream& operator<<(std::ostream& os, const file& f);

```

file.cpp

```
#include "file.h"
```

```

size_t_t file::get_x() const {
    return x;
}
size_t_t file::get_size() const {
    return size;
}
size_t_t file::get_index() const {
    return index;
}
bool file::get_sys() const {
    return opt.get_sys();
}
bool file::get_hid() const {
    return opt.get_hid();
}
std::string file::get_name() const {
    return name;
}
void file::change_x(const size_t_t& new_x) {
    x = new_x;
}
void file::change_size(const size_t_t& sz) {
    size = sz;
}
void file::change_index(const size_t_t& in) {
    index = in;
}
void file::change_sys(const bool& sys) {
    opt.change_sys(sys);
}
void file::change_hid(const bool& hid) {
    opt.change_hid(hid);
}
void file::change_name(const std::string& nm) {
    name = nm;
}
file::file() {
    x = 0;
    size = 100;
    index = 0;
    opt.change_hid(false);
}

```

```

    opt.change_sys(false);
    name = "File";
    std::cout << "Файл создан при помощи конструктора по умолчанию." << "\n";
}
file::file(const file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    opt = f.opt;
    name = f.name;
}
file::file(const size_t& ver, const size_t& sz, const size_t& ind, const bool& sys, const
bool& hid, const std::string& nm) {
    x = ver;
    size = sz;
    index = ind;
    opt.change_sys(sys);
    opt.change_hid(hid);
    name = nm;
    std::cout << "Файл создан при помощи конструктора с аргументами." << "\n";
}
file::~file() {
    std::cout << "Файл уничтожен при помощи деструктора по умолчанию." << "\n";
}
bool options::get_sys() const {
    return issystem;
}
bool options::get_hid() const {
    return ishidden;
}
void options::change_sys(const bool& sys) {
    issystem = sys;
}
void options::change_hid(const bool& hid) {
    ishidden = hid;
}
size_t executable_file::get_time() const {
    return runningtime;
}
executable_file::executable_file() : file(), runningtime(0) {
    type_of_file = 1;
}
executable_file::executable_file(const executable_file& sf) : file(sf), runningtime(sf.runningtime)
{
    type_of_file = 1;
}
executable_file::executable_file(const size_t& x, const size_t& sz, const size_t& ind, const
bool& sys, const bool& hid, const std::string& nm, const size_t& time) : file(x, sz, ind, sys,
hid, nm), runningtime(time) {
    type_of_file = 1;
}
executable_file::~executable_file() {}
void executable_file::change_time(const size_t& time) {
    runningtime = time;
}
bool non_executable_file::get_text() const {
    return is_text;
}
non_executable_file::non_executable_file() : file(), is_text(true) {
    type_of_file = 2;
}
non_executable_file::non_executable_file(const non_executable_file& nf) : file(nf),
is_text(nf.is_text) {
    type_of_file = 2;
}
non_executable_file::non_executable_file(const size_t& x, const size_t& sz, const size_t& ind,
const bool& sys, const bool& hid, const std::string& nm, const bool& text) : file(x, sz, ind, sys,
hid, nm), is_text(text) {
    type_of_file = 2;
}

```

```

}
non_executable_file::~non_executable_file() {}
void non_executable_file::change_text(const bool& text) {
    is_text = text;
}
std::string executable_file::get_info() const {
    std::stringstream s;
    s << type_of_file << " " << "\"" << name << "\"" << index << " " << size << " " << x << " "
<< opt.get_hid() << " " << opt.get_sys() << " " << runningtime;
    return s.str();
}
std::string non_executable_file::get_info() const {
    std::stringstream s;
    s << type_of_file << " " << "\"" << name << "\"" << index << " " << size << " " << x << " "
<< opt.get_hid() << " " << opt.get_sys() << " " << is_text;
    return s.str();
}
bool operator==(const file& f1, const file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const file& f1, const file& f2) {
    return !(f1 == f2);
}
bool operator==(const executable_file& f1, const executable_file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else if (f1.get_time() != f2.get_time()) {
        return false;
    }
    else {
        return true;
    }
}
}

```



```

bool operator!=(const executable_file& f1, const executable_file& f2) {
    return !(f1 == f2);
}
bool operator==(const non_executable_file& f1, const non_executable_file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else if (f1.get_text() != f2.get_text()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const non_executable_file& f1, const non_executable_file& f2) {
    return !(f1 == f2);
}
bool check_str(const std::string& str){
    std::regex re("[A-Za-zA-Яa-я0-9\\s\\!,\\?\\\"\\.\\.;\\']*");
    if (!(std::regex_search(str, re))) {
        return false;
    }
    std::regex re_2("\\s{2,}");
    if (std::regex_search(str, re_2)) {
        return false;
    }
    std::regex re_3("[\\!\\?\\.\\.,\\;]{2,}");
    if (std::regex_search(str, re_3)) {
        return false;
    }
    std::regex re_4("[\\'\\\"]{2,}");
    if (std::regex_search(str, re_4)) {
        return false;
    }
    return true;
}
std::ostream& operator<<(std::ostream& os, const file& f) {
    return os << f.get_info();
}
std::ostream& operator<<(std::ostream& os, const executable_file& f) {
    return os << f.get_info();
}
std::ostream& operator<<(std::ostream& os, const non_executable_file& f) {
    return os << f.get_info();
}
std::istream& operator>>(std::istream& is, executable_file& f) {
    std::string name;
    std::string temp;
    std::regex re("\\$");
    std::stringstream temps;
    executable_file tempf;
    bool check = true;
    bool global_check = true;
    do {

```

```

        is >> temp;
        if (check_str(temp)) {
            name += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(name, re)) {
            check = false;
        }
        else {
            name += " ";
        }
    } while (check);
    std::regex r("\\");
    name = std::regex_replace(name, r, "");
    tempf.change_name(name);
    int temp_i = 0;
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_index(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_size(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_x(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_hid(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_sys(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_time(temp_i);
    if (global_check == true) {
        f = tempf;
    }
}

```

```

        else {
            f.type_of_file = -1;
        }
        return is;
    }
}

std::istream& operator>>(std::istream& is, non_executable_file& f) {
    std::string name;
    std::string temp;
    std::regex re("\\$");
    std::stringstream temps;
    non_executable_file tempf;
    bool check = true;
    bool global_check = true;
    do {
        is >> temp;
        if (check_str(temp)) {
            name += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(name, re)) {
            check = false;
        }
        else {
            name += " ";
        }
    } while (check);
    std::regex r("\\");
    name = std::regex_replace(name, r, "");
    tempf.change_name(name);
    int temp_i = 0;
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_index(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_size(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_x(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_hid(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;

```

```

    temps >> temp_i;
    temps.clear();
    tempf.change_sys(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_text(temp_i);
    if (global_check == true) {
        f = tempf;
    }
    else {
        f.type_of_file = -1;
    }
    return is;
}
file& file::operator=(const file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    return *this;
}
executable_file& executable_file::operator=(const executable_file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    runningtime = f.runningtime;
    return *this;
}
non_executable_file& non_executable_file::operator=(const non_executable_file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    is_text = f.is_text;
    return *this;
}
std::string file::get_info() const {
    std::stringstream s;
    s << type_of_file << " " << "\'" << name << "\' " << index << " " << size << " " << x << " "
<< opt.get_hid() << " " << opt.get_sys();
    return s.str();
}
bool operator>(const file& f1, const file& f2) {
    return f1.get_name() < f2.get_name();
}
bool operator<(const file& f1, const file& f2) {
    return f1.get_name() > f2.get_name();
}
bool operator>(const executable_file& f1, const executable_file& f2) {
    return f1.get_name() < f2.get_name();
}
bool operator<(const executable_file& f1, const executable_file& f2) {
    return f1.get_name() > f2.get_name();
}
bool operator>(const non_executable_file& f1, const non_executable_file& f2) {
    return f1.get_name() < f2.get_name();
}
}

```

```
bool operator<(const non_executable_file& f1, const non_executable_file& f2) {
    return f1.get_name() > f2.get_name();
}
```

Mypointer.h

```
#pragma once
```

```
#include "file.h"
```

```
template <typename t> class My_ptr{
private:
    t* ptr;
    int* count;
public:
    My_ptr(t* ptr) {
        this->ptr = ptr;
        count = new int;
        *count = 1;
    }
    My_ptr(My_ptr& other) : ptr(other.ptr), count(other.count) {
        *count += 1;
    }
    ~My_ptr() {
        *count -= 1;
        if (*count == 0) {
            delete ptr;
            delete count;
        }
    }
    t* operator->() {
        return ptr;
    }
    t& operator*() {
        return *ptr;
    }
    My_ptr<t>& operator=(My_ptr& other) {
        *count -= 1;
        if (*count == 0) {
            delete ptr;
            delete count;
        }
        ptr = other.ptr;
        count = other.count;
        *count += 1;
        return *this;
    }
};
```

4. Результати роботи програми

Результати роботи програми:

```
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Данные из умных указателей переписаны в вектор.
0 "File" 0 100 0 0 0
Файл уничтожен при помощи деструктора по умолчанию.
0 "File" 0 100 0 0 0
Файл уничтожен при помощи деструктора по умолчанию.
0 "File" 0 100 0 0 0
Файл уничтожен при помощи деструктора по умолчанию.
0 "File" 0 100 0 0 0
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Утечка памяти не обнаружена.
```

5. Висновки

При виконанні даної лабораторної роботи було використано розумні вказівники з бібліотки STL і власний розумний вказівник. Незважаючи на те що пам'ять була виділена і не звільнена витоку пам'яті не відбулося, завдяки використанню розумних вказівників.

Програма протестована, витоків пам'яті немає, виконується без помилок.