Автор: Наймитенко С. КІТ-119а

Дата: 25 березня 2020

# **Лабораторна робота №6**

**СПАДКУВАННЯ**

Мета роботи: отримати знання про парадигму ООП – спадкування. Навчитися застосовувати отримані знання на практиці.

1 ЗАВДАННЯ ДО РОБОТИ

Модернізувати попередню лабораторну роботу шляхом: - додавання класу-спадкоємця, котрий буде поширювати функціонал «базового класу» відповідно до індивідуального завдання; - додавання ще одного класу-списку, що буде керувати лише елементами класу-спадкоємця; - у функціях базового класу та класу-спадкоємця обов’язкове використання ключових слів final та override.

Прикладна галузь – Компоненти програм

Базовий клас – Бібліотека, що підключається (модуль)

**Опис класів**

Базовий клас – C\_Library

Клас що має в собі динамічний масив базового класу та методи для роботи з ним – С\_List

Клас для зображення композиції – С\_Function

Клас для зображення агрегації – C\_Language

Клас наслідник базового класу C\_Library – С\_Expansion

Клас список класу наслідника С\_Expansion – C\_Heir\_list

**Опис змінних**

String expansion\_file – поле класу С\_Expansion(розширення файлу бібліотеки)

String language – поле класу C\_Language (назва мови програмування)

String function – поле класу С\_Function (назва функції з бібліотеки)

Int ID – поле класу C\_Library (ідентифікатор елемену)

Int year\_creating - – поле класу C\_Library (рік створення функції)

Int number\_of\_functions – поле класу C\_Library(кількість функцій в бібліотеці)

Int size – поле класу C\_List (змінна розміру масива)

C\_Library\* list – поле класу C\_Library (динамічний масив)

С\_List list\_lib - об’єкт класу С\_List

Int size – змінна розміру масиву

Int order – змінна індексу (порядкового номеру)

С\_Library new\_lib – змінна для нової бібліотеки

**Опис методів**

void Create() – метод класу C\_List (створення масиву і заповнення його данними)

void Add(C\_Library&,const int) - метод класу C\_List (додавання нового елементу в масив)

void Delete(const int) - метод класу C\_List (видалення елемента з масиву)

void Index\_output(const C\_Library,const int)const - метод класу C\_List (виведення на екран по індексу)

C\_Library Index\_return(const int) - метод класу C\_List (повернення елемента по індексу)

Void Output()const; - метод класу C\_List (виведення на екран)

Int Read\_file(string) - метод класу C\_List (зчитування з файлу)

void Count\_line(string) – метод класу C\_List (підрахунок кількості рядків у файлі)

C\_Library Distribution(string) - метод класу C\_List (розподіл , із рядка в об’єкт)

stringstream Str\_return(C\_Library&)const - метод класу C\_List (створення рядку змінних з об’єкта)

void Str\_output(stringstream&,int)const - метод класу C\_List (виведення рядка на екран)

float Difference() - метод класу C\_List (підрахуванн різнці між кількістю динамічно підключених бібліотек від іх загальної кількості)

int Write\_file(string) - метод класу C\_List (запис до файлу)

void Sort(func) - метод класу C\_List (сортування масиву елементів)

void If\_lib\_connected() - метод класу C\_List (виведення на екран данних про підключену бібліотеку (в якій мові програмування використовується, яку функцію має))

void What\_functions\_are(int) - метод класу C\_List ( шо виводить функцію з бібліотке на екран)

**Опис функцій**

С\_Library New\_Library() – створення одного елементу типу С\_Library та заповнення його данними з масиву

Bool Test\_Add() – тест методу додавання об’єкта

Bool Test\_Delete() – тест методу видалення об’єкта

Bool Test\_Index\_return() – тест методу повернення об’єкта по індексу

**Текст програми**

Файл Builder.h

#pragma once

#include "C\_List.h"

C\_Library New\_Lib(int);

Файл С\_Heir\_list.h

#pragma once

#include"C\_Library.h"

class C\_Heir\_list

{

private:

int size;

public:

C\_Expansion\* list;

void setSize\_array(const int size);

int getSize\_array()const;

int Read\_file(string);

void Output\_index(C\_Expansion)const;

void Output()const;

void Add(C\_Expansion,int);

void Delete(int);

};

Файл С\_Language.h

#pragma once

#include "C\_Library.h"

using std::string;

class C\_Language

{

private:

string language;

public:

void setLanguage\_programming(string) ;

void Which\_language\_programming();

C\_Language();

C\_Language(string);

C\_Language(const C\_Language&);

~C\_Language();

};

Файл C\_Library.h

#pragma once

#define CRTDBG\_MAP\_ALLOC

#include <crtdbg.h>

#define DEBUG\_NEW new(\_NORMAL\_BLOCK, FILE, \_\_LINE)

#include <iostream>

#include <clocale>

#include <ctime>

#include <cstring>

#include <iomanip>

#include <fstream>

#include <sstream>

#include <regex>

#include <Windows.h>

#include <stddef.h>

#include <cstddef>

#include "C\_Language.h"

using std::cin;

using std::cout;

using std::string;

using std::endl;

using std::setw;

using std::setprecision;

using std::ifstream;

using std::ofstream;

using std::getline;

using std::stringstream;

using std::getline;

using std::istringstream;

using std::regex;

using std::regex\_match;

using std::regex\_search;

typedef bool (func)(int, int);

class C\_Library

{

protected:

string dynamically;

string name;

int ID;

int year\_creating;

int number\_of\_functions;

class C\_Function final

{

private:

string function;

public:

void setFunction(string);

string What\_function\_is\_in\_this\_library();

C\_Function();

C\_Function(string);

C\_Function(const C\_Function&);

~C\_Function();

};

C\_Function function;

C\_Language lang;

public:

void setDynamically(string);

void setName(string);

void setID(const int);

void setYear\_Creating(const int);

void setNumber\_of\_function(const int);

void setLanguage\_programming(string);

void setFunction(string);

string getDynamically()const;

string getName()const;

int getID()const;

int getYear\_Creating()const;

int getNumber\_of\_function()const;

string What\_function\_is\_in\_this\_library();

string Which\_language\_programming();

C\_Library();

C\_Library(string, string, const int, const int, const int, C\_Library);

C\_Library(const C\_Library&);

~C\_Library();

};

typedef C\_Library\* arr\_Lib;

class C\_Expansion final : public C\_Library

{

private:

string expansion\_file;

public:

void setExpansion\_file(string);

string getExpansion\_file();

C\_Expansion();

C\_Expansion(string, C\_Library);

C\_Expansion(const C\_Expansion&);

~C\_Expansion();

};

Файл C\_List.h

#pragma once

#include "C\_Library.h"

#include"C\_Heir\_list.h"

class C\_List

{

private:

int size;

public:

arr\_Lib list;

C\_Heir\_list obj;

void setSize\_array(const int size);

int getSize\_array()const;

int Read\_file(string, string);//

void Count\_line(string);//

C\_Library Distribution(string, string);//

C\_Expansion\* Create();//

void Add(C\_Library, const int);//

void Delete(const int, C\_Expansion\*);//

stringstream Str\_return(C\_Library&)const;//

void Str\_output(stringstream&, int)const;//

C\_Library& Index\_return(const int index);//

void Output(C\_Expansion\*)const;//

void Sort(func,C\_Expansion\*&);

float Difference();

void If\_lib\_connected(C\_Expansion\*);

int Write\_file(string,C\_Expansion\*);

void Check(C\_Expansion\*);

~C\_List();

};

Файл Test.h

#pragma once

bool Test\_Add(C\_List&);

bool Test\_Delete(C\_List&);

bool Test\_Index\_return(C\_List&);

bool Test\_Difference(C\_List&);

Файл Builder.cpp

#include "Builder.h"

#include"C\_Library.h"

C\_Expansion New\_Lib(int i) // 2

{

int a = 0;

C\_Library new\_l;

int x;

srand(static\_cast<unsigned int>(time(0)));

if (i % 2 == 0)

x = rand() % 1;

else

x = 2;

if (x == 0)

{

string dynamically = "yes";

string name = "math" + i;

string func = "Default";

string lang = "default";

string expansion = ".dll";

new\_l.setFunction(func);

new\_l.setLanguage\_programming(lang);

C\_Library new\_lib(dynamically, name, 20 + i, 1998, 40, new\_l);

C\_Expansion new\_el(expansion, new\_lib);

return new\_el;

}

else if (x == 1)

{

string dynamically = "no";

string name = "string" + i;

string func = "Default";

string lang = "default";

string expansion = ".dll";

new\_l.setFunction(func);

new\_l.setLanguage\_programming(lang);;

C\_Library new\_lib(dynamically, name, 46 + i, 2000, 20, new\_l);

C\_Expansion new\_el(expansion, new\_lib);

return new\_el;

}

else if (x == 2)

{

string expansion = ".dll";

C\_Library new\_lib(new\_l);

int id = new\_lib.getID();

new\_lib.setID(id + i);

C\_Expansion new\_el(expansion, new\_lib);

return new\_el;

}

}

Файл С\_Heir\_list.cpp

#include "C\_Heir\_list.h"

void C\_Heir\_list::setSize\_array(const int size)

{

this->size = size;

}

int C\_Heir\_list::getSize\_array()const

{

return this->size;

}

int C\_Heir\_list::Read\_file(string file\_name)

{

ifstream file(file\_name);

if (!file)

{

cout << "Ошибка!!! Файл не открыто." << endl;

return 1;

}

string line;

for (int i = 0; i < size; i++)

{

file >> line;

list[i].setExpansion\_file(line);

}

file.close();

return 0;

}

void C\_Heir\_list::Output\_index(C\_Expansion el)const

{

cout << setw(17)<< el.getExpansion\_file() << endl;

}

void C\_Heir\_list::Output()const

{

cout << "Вывод на экран разширений файлов" << endl;

for (int i = 0; i < size; i++)

Output\_index(list[i]);

}

void C\_Heir\_list::Add(C\_Expansion new\_el,int order)

{

size++;

C\_Expansion\* new\_list = new C\_Expansion[size];

for (size\_t i = 0, j = 0; i < size; i++)

{

if (i != order - 1)

{

new\_list[i] = list[j];

j++;

}

else

new\_list[i] = new\_el;

}

delete[] list;

list = new C\_Expansion[size];

for (size\_t i = 0; i < size; i++)

list[i] = new\_list[i];

delete[] new\_list;

}

void C\_Heir\_list::Delete(int order)

{

size--;

C\_Expansion\* new\_list = new C\_Expansion[size];

for (size\_t i = 0; i < order; i++)

new\_list[i] = list[i];

for (size\_t i = order; i < size; i++)

new\_list[i] = list[i + 1];

delete[] list;

list = new C\_Expansion[size];

for (size\_t i = 0; i < size; i++)

list[i] = new\_list[i];

delete[] new\_list;

}

Файл С\_Language.cpp

#include"C\_Library.h"

#include"C\_Language.h"

void C\_Language::setLanguage\_programming(string name)

{

language = name;

}

void C\_Language::Which\_language\_programming()

{

cout << language ;

}

C\_Language::C\_Language() :language("Default")

{

}

C\_Language::C\_Language(string language) : language(language)

{

}

C\_Language::C\_Language(const C\_Language& lang) : language(lang.language)

{

}

C\_Language::~C\_Language()

{

}

Файл C\_Library.cpp

#include "C\_Library.h"

void C\_Library::C\_Function::setFunction(string name)

{

this->function = name;

}

string C\_Library::C\_Function::What\_function\_is\_in\_this\_library() //дилигирование

{

return function;

}

void C\_Library::setDynamically(string dynamically)

{

this->dynamically = dynamically;

}

void C\_Library::setName(string name)

{

this->name = name;

}

void C\_Library::setID(const int ID)

{

this->ID = ID;

}

void C\_Library::setYear\_Creating(const int year\_creating)

{

this->year\_creating = year\_creating;

}

void C\_Library::setNumber\_of\_function(const int number\_of\_functions)

{

this->number\_of\_functions = number\_of\_functions;

}

void C\_Library::setLanguage\_programming(string lang)

{

this->lang.setLanguage\_programming(lang);

}

void C\_Library::setFunction(string name)

{

function.setFunction(name);

}

string C\_Library::getDynamically()const

{

return dynamically;

}

string C\_Library::getName()const

{

return name;

}

int C\_Library::getID()const

{

return this->ID;

}

int C\_Library::getYear\_Creating()const

{

return this->year\_creating;

}

int C\_Library::getNumber\_of\_function()const

{

return this->number\_of\_functions;

}

string C\_Library::What\_function\_is\_in\_this\_library() //дилигирование

{

return function.What\_function\_is\_in\_this\_library();

}

string C\_Library::Which\_language\_programming()

{

return lang.Which\_language\_programming();

}

C\_Library::C\_Function::C\_Function() : function("Default")

{

}

C\_Library::C\_Function::C\_Function(string function) : function(function)

{

}

C\_Library::C\_Function::C\_Function(const C\_Function& func) : function(func.function)

{

}

C\_Library::C\_Function::~C\_Function()

{

}

C\_Library::C\_Library() :dynamically("yes"), name("Default"), ID(10), year\_creating(1999), number\_of\_functions(50), function("default"), lang("default")

{

}

C\_Library::C\_Library(string dynamically, string name, const int ID, const int year\_creating, const int number\_of\_functions, C\_Library lib) : dynamically(dynamically), name(name), ID(ID), year\_creating(year\_creating), number\_of\_functions(number\_of\_functions), function(lib.function), lang(lib.lang)

{

}

C\_Library::C\_Library(const C\_Library& lib) : dynamically(lib.dynamically), name(lib.name), ID(lib.ID), year\_creating(lib.year\_creating), number\_of\_functions(lib.number\_of\_functions), function(lib.function), lang(lib.lang)

{

}

C\_Library::~C\_Library()

{

}

void C\_Expansion::setExpansion\_file(string expansion)

{

this->expansion\_file = expansion;

}

string C\_Expansion::getExpansion\_file()

{

return this->expansion\_file;

}

C\_Expansion::C\_Expansion() : expansion\_file("Default"), C\_Library::C\_Library()

{

}

C\_Expansion::C\_Expansion(string expansion\_file, C\_Library cop\_el) : expansion\_file(expansion\_file), C\_Library::C\_Library(cop\_el.getDynamically(), cop\_el.getName(), cop\_el.getID(), cop\_el.getYear\_Creating(), cop\_el.getNumber\_of\_function(), cop\_el)

{

}

C\_Expansion::C\_Expansion(const C\_Expansion& elem) : expansion\_file(elem.expansion\_file), C\_Library::C\_Library(elem.getDynamically(), elem.getName(), elem.getID(), elem.getYear\_Creating(), elem.getNumber\_of\_function(), elem)

{

}

C\_Expansion::~C\_Expansion()

{

}

Файл C\_List.cpp

#include "C\_List.h"

#include "Builder.h"

#include"C\_Language.h"

#include"C\_Library.h"

void C\_List::setSize\_array(const int size)

{

this->size = size;

}

int C\_List::getSize\_array()const

{

return this->size;

}

int C\_List::Read\_file(string file\_name, string file\_name2)

{

ifstream file(file\_name);

if (!file)

{

cout << "Ошибка!!! Файл не открыто." << endl;

return 1;

}

ifstream file2(file\_name2);

if (!file2)

{

cout << "Ошибка!!! Файл не открыто." << endl;

return 1;

}

string line;

string line2;

for (size\_t i = 0; i < size; i++)

{

getline(file, line);

getline(file2, line2);

list[i] = Distribution(line, line2);

}

file.close();

return 0;

}

void C\_List::Count\_line(string file\_name)

{

int size2 = 0;

size = size2;

ifstream file(file\_name);

if (!file)

{

cout << "Ошибка!!! Файл не открыто." << endl;

return;

}

string line;

while (getline(file, line))

{

size2++;

}

size = size2;

file.close();

}

C\_Library C\_List::Distribution(string line, string line2)

{

regex regular("([\\d]\* [\\d]\* [\\d]\* (Yes|No) [A-ZА-Я]+[\\wА-Яа-я\-|\_|:|;|\.]\*[ ]?([A-ZА-Я]+[\\wА-Яа-я\-|\_|:|;|\.]\*)?)");

regex replace\_reg1("([;]{2,})");

regex replace\_reg2("([\_]{2,})");

regex replace\_reg3("([-]{2,})");

regex replace\_reg4("([:]{2,})");

regex replace\_reg5("([\.]{2,})");

C\_Library new\_l;

int id, number\_of\_function, year\_creating;

string dynamically, name = " ", name2 = " ", line\_res;

string name\_f, name\_l;

auto res = regex\_match(line, regular);

if (res)

{

if (regex\_search(line, replace\_reg1))

{

line = regex\_replace(line, replace\_reg1, ";");

}

if (regex\_search(line, replace\_reg2))

{

line = regex\_replace(line, replace\_reg2, "\_");

}

if (regex\_search(line, replace\_reg3))

{

line = regex\_replace(line, replace\_reg3, "-");

}

if (regex\_search(line, replace\_reg4))

{

line = regex\_replace(line, replace\_reg4, ":");

}

if (regex\_search(line, replace\_reg5))

{

line = regex\_replace(line, replace\_reg5, ".");

}

istringstream str(line);

str >> id >> number\_of\_function >> year\_creating >> dynamically >> name >> name2;

if (name2 != " " && name != " ")

name = name + " " + name2;

istringstream str2(line2);

str2 >> name\_f >> name\_l;

new\_l.setFunction(name\_f);

new\_l.setLanguage\_programming(name\_l);

C\_Library new\_el(dynamically, name, id, year\_creating, number\_of\_function, new\_l);

return new\_el;

}

C\_Library new\_el;

return new\_el;

}

C\_Expansion\* C\_List::Create() // 1

{

cout << "Происходит заполнеие масива данными" << endl;

C\_Expansion\* list\_e = new C\_Expansion[size];

for (size\_t i = 0; i < size; i++)

{

list\_e[i] = New\_Lib(i + 1);

list[i].setDynamically(list\_e[i].getDynamically());

list[i].setID(list\_e[i].getID());

list[i].setName(list\_e[i].getName());

list[i].setLanguage\_programming(list\_e[i].Which\_language\_programming());

list[i].setFunction(list\_e[i].What\_function\_is\_in\_this\_library());

list[i].setYear\_Creating(list\_e[i].getYear\_Creating());

list[i].setNumber\_of\_function(list\_e[i].getNumber\_of\_function());

}

return list\_e;

}

void C\_List::Add(C\_Library lib, const int order) // 3

{

size++;

arr\_Lib new\_list = new C\_Library[size];

for (size\_t i = 0, j = 0; i < size; i++)

{

if (i != order - 1)

{

new\_list[i] = list[j];

j++;

}

else

new\_list[i] = lib;

}

delete[] list;

list = new C\_Library[size];

for (size\_t i = 0; i < size; i++)

list[i] = new\_list[i];

delete[] new\_list;

}

void C\_List::Delete(const int order,C\_Expansion\* list\_e) // 4

{

size--;

arr\_Lib new\_list = new C\_Library[size];

cout << "Елемент который вы удалили" << endl;

stringstream str = Str\_return(list[order - 1]);

Str\_output(str, 0);

obj.Output\_index(list\_e[order-1]);

for (size\_t i = 0; i < order - 1; i++)

new\_list[i] = list[i];

for (size\_t i = order - 1; i < size; i++)

new\_list[i] = list[i + 1];

delete[] list;

list = new C\_Library[size];

for (size\_t i = 0; i < size; i++)

list[i] = new\_list[i];

delete[] new\_list;

}

C\_Library& C\_List::Index\_return(const int index)

{

int id;

for (size\_t i = 0; i < size; i++)

{

id = list[i].getID();

if (id == index)

{

id = i;

i = size;

}

}

return list[id];

}

stringstream C\_List::Str\_return(C\_Library& lib)const // 5

{

stringstream str;

str << " " << lib.getID() << " " << lib.getNumber\_of\_function() << " " << lib.getYear\_Creating() << " " << lib.getDynamically() << " " << lib.getName();

return str;

}

void C\_List::Str\_output(stringstream& str, int i)const

{

int number\_of\_function, year\_creating, id;

string dynamically, name1, name2 = "";

str >> id;

str >> number\_of\_function;

str >> year\_creating;

str >> dynamically;

str >> name1;

str >> name2;

if (name2 == "")

cout << i + 1 << "- " << setw(10) << id << setw(13) << number\_of\_function << setw(21) << year\_creating << setw(20) << dynamically << setw(29) << name1;

else

cout << i + 1 << "- " << setw(10) << id << setw(13) << number\_of\_function << setw(21) << year\_creating << setw(20) << dynamically << setw(23) << name1 << " " << name2;

}

void C\_List::Output(C\_Expansion\* list\_e)const // 6

{

stringstream str;

cout << "Вивод на экран всех объектов" << endl;

cout.setf(std::ios::right);

cout << "№" << setw(17) << "id библиотеки" << setw(16) << "Кол-во функций" << setw(18) << "Год её создания" << setw(30) << "Динмически ли она подключена" << setw(20) << "Название библиотеки" << setw(29) << "Расширение файла библиоткеки" << endl;

for (size\_t i = 0; i < size; i++)

{

str = Str\_return(list[i]);

Str\_output(str, i);

obj.Output\_index(list\_e[i]);

}

}

void C\_List::Sort(func condition,C\_Expansion\*& list\_e)

{

C\_Library temp;

C\_Expansion temp\_e;

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

if (condition(list[i].getID(), list[j].getID()))

{

temp = list[i];

temp\_e = list\_e[i];

list[i] = list[j];

list\_e[i] = list\_e[j];

list[j] = temp;

list\_e[j] = temp\_e;

}

}

}

}

float C\_List::Difference()

{

int count = 0;

string y;

for (size\_t i = 0; i < size; i++)

{

y = list[i].getDynamically();

if (y == "yes")

count++;

}

float dif = (float)size / (float)count;

cout << "В " << setprecision(5) << dif << " раз количество библиотек, которые динамически подключаются, меньше чем общее количество библиотек" << endl;

return dif;

}

void C\_List::If\_lib\_connected(C\_Expansion\* list\_e)

{

for (size\_t i = 0; i < size; i++)

{

cout << endl << "В языке програмирования \"";

cout << list[i].Which\_language\_programming();

cout << "\" используется библиотека " << list[i].getName() << endl;

cout << "В данной библиотеке есть такая функция" << endl;

cout << "Функция - " << list[i].What\_function\_is\_in\_this\_library() << endl;

cout << "Расширения библиотеки - " << list\_e[i].getExpansion\_file() << endl;

}

}

int C\_List::Write\_file(string file\_name, C\_Expansion\* list\_e)

{

ofstream file(file\_name);

if (!file)

{

cout << "Ошибка!!! Файл не открыто." << endl;

return 1;

}

file << "№" << setw(17) << "id библиотеки" << setw(16) << "Кол-во функций" << setw(18) << "Год её создания" << setw(30) << "Динмически ли она подключена" << setw(17) << "Название библиотеки" << setw(17) << "Расширение файла библиоткеки" << endl;

for (size\_t i = 0; i < size; i++)

file << i + 1 << "- " << setw(5) << list[i].getID() << setw(24) << list[i].getNumber\_of\_function() << setw(27) << list[i].getYear\_Creating() << setw(23) << list[i].getDynamically() << setw(23) << list[i].getName() << setw(23) << list\_e[i].getExpansion\_file()<< endl;

file.close();

}

void C\_List::Check(C\_Expansion\* list\_e)

{

regex regular("(([A-ZА-Я]+[\\wА-Яа-я\-|\_|:|;|\.]\*)+( )+([A-ZА-Я]+[\\wА-Яа-я\-|\_|:|;|\.]\*))");

string check;

stringstream str;

int k = 0;

for (size\_t i = 0; i < size; i++)

{

auto check\_res = list[i].getName();

if (regex\_match(check\_res, regular))

{

str = Str\_return(list[i]);

Str\_output(str, k);

obj.Output\_index(list\_e[i]);

k++;

}

}

}

C\_List:: ~C\_List()

{

delete[] list;

}

Файл Test.cpp

#include "C\_Library.h"

#include "C\_List.h"

#include "Test.h"

int main()

{

setlocale(LC\_ALL, "Rus");

bool result;

C\_List list;

list.list = new C\_Library[3];

list.setSize\_array(3);

result = Test\_Add(list);

if (result)

cout << "Test\_Add пройден успешно!" << endl;

else

cout << "Test\_Add не пройден!" << endl;

result = Test\_Delete(list);

if (result)

cout << "Test\_Delete пройден успешно!" << endl;

else

cout << "Test\_Delete не пройден!" << endl;

result = Test\_Index\_return(list);

if (result)

cout << "Test\_Index\_return пройден успешно!" << endl;

else

cout << "Test\_Index\_return не пройден!" << endl;

result = Test\_Difference(list);

if (result)

cout << "Test\_Difference пройден успешно!" << endl;

else

cout << "Test\_Difference не пройден!" << endl;

return 0;

}

Файл Test\_method.cpp

#include "C\_Library.h"

#include "C\_List.h"

#include "Test.h"

bool Test\_Add(C\_List& list)

{

C\_Library new\_lib;

list.Add(new\_lib, 4);

int size = list.getSize\_array();

if (size == 4)

return true;

else

return false;

}

bool Test\_Delete(C\_List& list)

{

int size = list.getSize\_array();

C\_Expansion\* obj = new C\_Expansion[size];

list.Delete(1,obj);

size = 0;

size = list.getSize\_array();

if (size == 3)

return true;

else

return false;

}

bool Test\_Index\_return(C\_List& list)

{

string dynamically = "yes";

string name = "Default", func = "func", lang = "lang";

C\_Library n\_el;

n\_el.setFunction(func);

n\_el.setLanguage\_programming(lang);

C\_Library new\_el(dynamically, name, 20, 2001, 29, n\_el);

list.list[0] = new\_el;

C\_Library return\_lib = list.Index\_return(20);

int count = 0, value = 0;

string y1 = return\_lib.getDynamically();

string y2 = "yes";

if (y1 == y2)

count++;

y1 = return\_lib.getName();

y2 = "Default";

if (y1 == y2)

count++;

value = return\_lib.getID();

if (value == 20)

count++;

value = return\_lib.getYear\_Creating();

if (value == 2001)

count++;

value = return\_lib.getNumber\_of\_function();

if (value == 29)

count++;

y1 = return\_lib.What\_function\_is\_in\_this\_library();

y2 = "func";

if (y1 == y2)

count++;

y1 = return\_lib.Which\_language\_programming();

y2 = "lang";

if (y1 == y2)

count++;

if (count == 7)

return true;

else

return false;

}

bool Test\_Difference(C\_List& list)

{

string dynamically = "yes";

string name = "Default", func = "Default", lang = "Default";

C\_Library n\_el;

n\_el.setFunction(func);

n\_el.setLanguage\_programming(lang);

C\_Library new\_lib1(dynamically, name, 20, 1998, 40, n\_el);

list.list[1] = new\_lib1;

C\_Library new\_lib2;

list.list[2] = new\_lib2;

float diff = list.Difference();

if (diff == 1.5)

return true;

else

return false;

}

Файл Main.cpp

#include "C\_List.h"

#include"C\_Heir\_list.h"

#include"C\_Library.h"

typedef bool (func)(int, int);

bool A\_more(int a, int b) { return a > b; }

bool B\_more(int a, int b) { return b > a; }

int Menu();

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Rus");

int res = Menu();

if (res == 0)

return 0;

if (\_CrtDumpMemoryLeaks())

cout << endl << "Есть утечка памяти" << endl;

else

cout << endl << "Нет утечки памяти" << endl;

return 0;

}

int Menu()

{

C\_List list\_lib;

C\_Heir\_list list\_exp;

cout << "Cоздание масива" << endl;

int command;

cout << "Введите вариант заполнения масива 1- из файла 2-конструктором" << endl;

cin >> command;

string file\_read1 = "file\_read.txt";

string file\_read2 = "expansion.txt";

string file\_func = "All\_func.txt";

string file\_write = "file\_write.txt";

int res;

if (command == 1)

{

list\_lib.Count\_line(file\_read1);

int size = list\_lib.getSize\_array();

list\_exp.setSize\_array(size);

list\_lib.list = new C\_Library[size];

list\_exp.list = new C\_Expansion[size];

res = list\_lib.Read\_file(file\_read1, file\_func);

if (res == 1)

return 1;

res = list\_exp.Read\_file(file\_read2);

if (res == 1)

return 1;

}

else

{

int size;

cout << "Введите размер масива" << endl;

cin >> size;

list\_lib.setSize\_array(size);

list\_lib.list = new C\_Library[size];

list\_exp.list = new C\_Expansion[size];

list\_exp.list = list\_lib.Create();

}

list\_lib.Output(list\_exp.list);

int order = 0;

cout << "Введите порядковый номер который вы хотите присвоить новому елементу" << endl;

cin >> order;

C\_Library new\_lib;

C\_Expansion new\_exp;

cout << "Добавления нового елемента" << endl;

list\_lib.Add(new\_lib, order);

list\_exp.Add(new\_exp, order);

list\_lib.Output(list\_exp.list);

cout << "Удаления елемента" << endl;

cout << "Введите номер елемента которого вы хотите удалить" << endl;

cin >> order;

cout << "Элемент который вы удалили" << endl;

list\_lib.Delete(order, list\_exp.list);

list\_lib.Output(list\_exp.list);

cout << "Введите индекс елемента данные которого вы хотите получить " << endl;

cin >> order;

C\_Library index\_lib = list\_lib.Index\_return(order);

cout << "Вывод полученого элемента на екран" << endl;

stringstream str = list\_lib.Str\_return(index\_lib);

list\_lib.Str\_output(str, 0);

list\_lib.Sort(A\_more,list\_exp.list);

list\_lib.Output(list\_exp.list);

list\_lib.Sort(B\_more, list\_exp.list);

list\_lib.Output(list\_exp.list);

cout << "Вывод на экран библиотек, в которых в названии 2 слова " << endl;

list\_lib.Check(list\_exp.list);

float diff = list\_lib.Difference();

cout << "Проверка. Какие библиотеки сейчас подключены." << endl;

list\_lib.If\_lib\_connected(list\_exp.list);

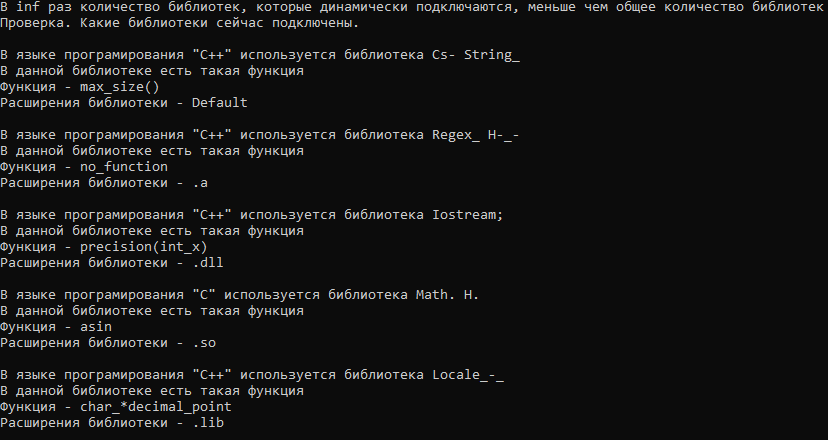
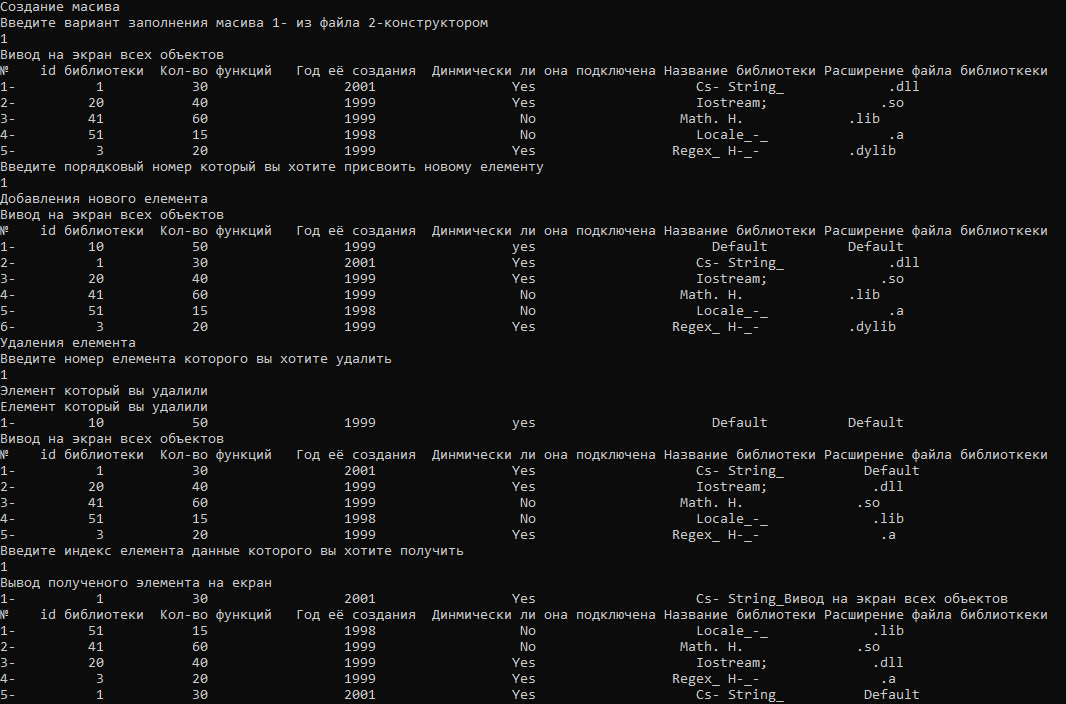
res = list\_lib.Write\_file(file\_write, list\_exp.list);

if (res == 1)

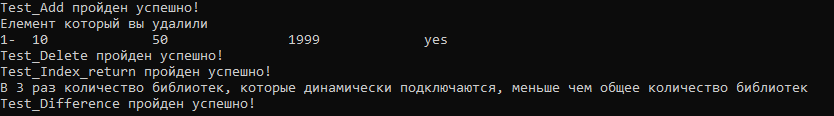
return 1;

}

4 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ



Результат роботи основоного завдання



Результат тестування методів

ВИСНОВКИ

Отримали знання про парадигму ООП – спадкування. Навчитися застосовувати отримані знання на практиці.