

Лабораторна робота 8. ПОЛІМОРФІЗМ

Тема. Перевантаження операторів. Серіалізація.

Мета: отримати знання про призначення операторів, визначити їх ролі у житті об'єкта та можливість перевизначення.

1. Завдання до роботи

Загальне завдання. Поширити попередню лабораторну роботу таким чином:

- у базовому класі, та класі/класах-спадкоємцях перевантажити:
 - оператор присвоювання;
 - оператор порівняння (на вибір: == , < , > , >= , <= , !=);
 - оператор введення / виведення;
- у класі-списку перевантажити:
 - оператор індексування ([]);
 - оператор введення / виведення з акцентом роботи, у тому числі і з файлами. При цьому продовжувати використовувати регулярні вирази для валідації введених даних.

2.1. Опис змінних

Базовий абстрактний клас: [CProgram](#).

Клас, що має в собі масиви класів-спадкоємців та методи для роботи з ними: [CList](#).

Клас, що відображає агрегативні відносини з базовим класом: [CAuthor](#).

Клас, що відображає ком позитивні відносини з базовим класом: [CDate](#).

Клас-спадкоємець: [CMalware](#).

Клас-спадкоємець: [CProgramForSale](#).

2.2. Опис змінних

[int](#) timeOfWork – поле класу [CProgram](#) (час виконання програми).

[int](#) size – поле класу [CProgram](#) (розмір програми у мегабайтах).

[int](#) amountOfLines – поле класу [CProgram](#) (кількість рядків коду).

[int](#) index – поле класу [CProgram](#) (ідентифікаційний номер).

[bool](#) useInternet – поле класу [CProgram](#) (використовує інтернет).

[string](#) name – поле класу [CProgram](#) (назва програми).

[CAuthor](#) author – поле класу [CAuthor](#) (автор програми).

[CDate](#) date – поле класу [CDate](#) (дата створення програми).

[int](#) listSize – поле класу [CList](#) (розмір масиву елементів класу [CProgram](#)).

[C_Program**](#) programList – поле класу [CProgram](#) (масив елементів класу [CProgram](#)).

[sint](#) day, month, year – поле класу [CDate](#) (дата).

[string](#) type – поле класу [CMalware](#) (тип зловмисного ПО).

`int price` – поле класу `CProgramForSale` (ціна).

2.3. Опис методів

`int getListSize() const` – отримання даних змінної розміру масиву елементів класу `Program` (метод класу `CList`).

`void createList(int)` – створення масиву елементів і заповнення даними (метод класу `CList`).

`void printAll() const` – виведення даних елементів у консоль (метод класу `CList`).

`void addProgram(CProgram*)` – додавання нового елемента в масив (метод класу `CList`).

`void delProgram(int)` – видалення елемента з масиву (метод класу `CList`).

`sint task(int) const` – знаходження елементів за певним критерієм (метод класу `CList`).

`int getProgramID(int) const` – отримання даних елемента по індексу (метод класу `CList`).

`int linesInFile(string) const` – знаходження кількості рядків у файлі (метод класу `CList`).

`void readFile(string)` – читання даних з файлу (метод класу `CList`).

`void getOneEl(int) const` – отримання строки даних (метод класу `CList`).

`void saveFile(string) const` – збереження даних у файл (метод класу `CList`).

`void enterNewEl()` – введення даних з клавіатури (метод класу `CList`).

`int regexTask() const` – виведення елементів, назва яких містить 2 слова (метод класу `CList`).

`void sort(comp) const` – функція сортування списку елементів (метод класу `CList`).

`~CList()` – деструктор списку елементів (метод класу `CList`).

`CProgram()` – конструктор без параметра (метод класу `CProgram`).

`CProgram(bool, int, int, int, int, string, CAuthor, sint, sint, sint)` – конструктор класу з параметрами (метод класу `CProgram`).

`CProgram(const C_Program& other)` – конструктор копіювання (метод класу `CProgram`).

`~CProgram()` – деструктор елемента (метод класу `CProgram`).

2.4 Опис функцій

`void Menu()` – функція меню.

`void Test_GetProgramID(CList&)` – тест функції знаходження та повернення об'єкту по індексу.

`void Test_AddEl(CList&, CAuthor*)` – тест функції додавання об'єкта до масиву об'єктів.

`void Test_DelEl(CList&)` – тест функції видалення об'єкта з масиву об'єктів.

`void Test_Task(CList&)` – тест функції знаходження елементів за певними критеріями(індивідуальне завдання).

`void Test_Stringstream(CList&)` – тест функції отримання даних зі строки.

`void Test_ReadFile(CList&)` – тест функції читання даних з файлу.

`void Test_RegexTask(CList&)` – тест функції отримання даних програм, які містять 2 слова.

`void Test_Sort(CList&)` – тест функції сортування списку.

`void Test_Aggregation(CAuthor*)` – тест функції агрегативних відносин класів.

3. Текст програми test.cpp

```
#include "programList.h"
```

```
void Test_GetProgramID(CList&);  
void Test_AddEl(CList&, CAuthor*);  
void Test_DelEl(CList&);  
void Test_Task(CList&);  
void Test_Stringstream(CList&);  
void Test_ReadFile(CList&);  
void Test_RegexTask(CList&);  
void Test_Sort(CList&);  
void Test_Aggregation(CAuthor*);  
void funcTest();
```

```
int main()  
{  
    setlocale(LC_ALL, "Rus");  
  
    funcTest();  
  
    if (_CrtDumpMemoryLeaks())  
        cout << "\nЕсть утечка памяти.\n";  
    else cout << "\nУтечка памяти отсутствует.\n";  
  
    return 0;  
}
```

```
void funcTest()  
{  
    CList list;  
    CAuthor author;  
    CAuthor* authorsList;  
  
    authorsList = author.createList(6);  
    list.createList(4, authorsList);  
  
    Test_GetProgramID(list);  
    Test_AddEl(list, authorsList);  
    Test_DelEl(list);  
    Test_Task(list);  
    Test_Stringstream(list);  
    Test_ReadFile(list);  
    Test_RegexTask(list);  
    Test_Sort(list);  
    Test_Aggregation(authorsList);  
}
```

```

}

void Test_GetProgramID(CList& list)
{
    int expected = 7896;
    int real = list[2]->getIndex();

    if(expected == real) cout << "Тест получения элемента по ID\t\t выполнен успешно.\n";
    else cout << "Тест получения элемента по ID\t\t не выполнен успешно.\n";
}

void Test_AddEl(CList& list, CAuthor* listAuthors)
{
    int value = list.getListSize();
    CProgram* newProgram = list.newProgram(listAuthors, 1);
    list.addProgram(newProgram);

    if (list.getListSize() > value)
        cout << "Тест добавления элемента в список\t выполнен успешно.\n";
    else cout << "Тест добавления элемента в список\t не выполнен успешно.\n";
}

void Test_DelEl(CList& list)
{
    int size = list.getListSize();
    list.delProgram(1);
    int newSize = list.getListSize();

    if (size > list.getListSize())
        cout << "Тест функции удаления\t\t\t выполнен успешно.\n\n";
    else cout << "Тест функции удаления\t\t\t не выполнен успешно.\n\n";
}

void Test_Task(CList& list)
{
    int expected = 0;
    int real = list.task(200);

    cout << endl;
    if(expected == real) cout << "Тест нахождения элементов по параметру\t выполнен успешно.\n";
    else cout << "Тест нахождения элементов по параметру\t не выполнен успешно.\n";
}

void Test_Stringstream(CList& list)
{
    string nameExpected = "1445";
    stringstream funcResult = list.testStringstream(1);
    string nameReal;
    funcResult >> nameReal;

    if (nameExpected == nameReal) cout << "Тест функции stringstream\t\t выполнен успешно." << endl;
    else cout << "Тест функции stringstream\t\t не выполнен успешно." << endl;
}

void Test_ReadFile(CList& list)
{
    int expected = 3;
    int real = list.linesInFile("data.txt");

    if (expected == real) cout << "Тест функции чтения из файла\t\t выполнен успешно." << endl;
    else cout << "Тест функции чтения из файла\t\t не выполнен успешно." << endl;
}

void Test_RegexTask(CList& list)
{
    int expected = 0;
    int real = list.regexTask();

    if (real == expected) cout << "Тест функции regex\t\t\t выполнен успешно." << endl;
    else cout << "Тест функции regex\t\t\t не выполнен успешно." << endl;
}

void Test_Sort(CList& list)
{

```

```

stringstream str1 = list.testStringstream(0);
string beforeSorting;
str1 >> beforeSorting;

list.sort(list.sortDesc);

stringstream str2 = list.testStringstream(0);
string afterSorting;
str2 >> afterSorting;

if (beforeSorting != afterSorting && afterSorting == "1445")
    cout << "Тест функции сортировки\t\t\t выполнен успешно." << endl;
else cout << "Тест функции сортировки\t\t\t не выполнен успешно." << endl;
}
void Test_Aggregation(CAuthor* list)
{
    string expected = "Lambda";
    string real = (list + 1)->getAuthor();

    if (expected == real) cout << "Тест агрегации\t\t\t выполнен успешно." << endl;
    else cout << "Тест агрегации\t\t\t не выполнен успешно." << endl;
}

```

main.cpp

```

#include "programList.h"

void menu();

int main()
{
    setlocale(LC_ALL, "Rus");
    menu();

    if (_CrtDumpMemoryLeaks()) cout << endl << "Есть утечка памяти." << endl;
    else cout << endl << "Утечка памяти отсутствует." << endl;

    return 0;
}

void menu()
{
    CProgram** programList;
    string fileName;           //название файла
    CList list;                //список программ
    CAuthor* listAuthors;      //список авторов
    CAuthor author;            //переменная поля автор
    auto chose = 1;
    auto chose2 = 0;
    auto chose3 = 0;
    auto value = 0, stop = 1;
    int result = 0, b, size;
    string::size_type n;
    stringstream str;

    listAuthors = author.createList(6);
    list.createList(4, listAuthors);

    cout << endl << "Выберите команду для работы со списком: ";
    while (stop != 0)
    {
        if (list.getListSize() == 0)
        {
            cout << "Список пуст. Что вы хотите сделать?" << endl;
            cout << "1) Добавить элемент вручную" << endl;
            cout << "2) Прочитать данные из файла" << endl;
            cout << "3) Завершение работы" << endl;

```

```

cout << "======" << endl;
cout << "Ваш выбор: ";
cin >> choise;
cout << endl;

switch (choise)
{
case 1:
    list.enterNewProgram();
    break;

case 2:
    cout << "Введите название файла для чтения данных для базового класса: ";
    cin >> fileName;
    cout << endl;

    n = fileName.find(".txt");
    if (n > 187) fileName += string(".txt");

    list.readFile(fileName);

    break;

case 3:
    cout << "Завершение работы." << endl;
    stop = 0;
    break;

default:
    cout << "Неверный номер элемента. Повторите попытку." << endl;
    break;
}
}
else
{
    cout << endl;
    cout << "1)Вывод на экран" << endl;
    cout << "2)Работа с файлами" << endl;
    cout << "3)Сортировка данных" << endl;
    cout << "4)Удаление элемента" << endl;
    cout << "5)Добавление элементов" << endl;
    cout << "6)Завершение работы" << endl;
    cout << "======" << endl;
    cout << "Ваш выбор: ";
    cin >> choise;
    cout << endl;
}

switch (choise)
{
case 1:
    cout << "Выберите команду:" << endl;
    cout << "1) Вывести весь список на экран" << endl;
    cout << "2) Вывести список программ больше определённого размера за авторством  
Microsoft" << endl;
    cout << "3) Вывести список программ, названия которых состоят из 2 слов" << endl;

    cout << "4) Вывести программу по ID" << endl;
    cout << "5) Вернуться к выбору действий" << endl;
    cout << "======" << endl;
    cout << "Ваш выбор: ";
    cin >> choise2;
    cout << endl;

    switch (choise2)
    {
    case 1:
        list.printAll();
        break;
    }
}
}

```

```

    case 2:
        cout << "Введите минимальный размер программ: ";
        cin >> value;
        cout << endl;

        size = list.task(value);
        break;

    case 3:
        list.regexTask();

        if (size == 0)
            cout << "Отсутствуют программы, содержащие 2 слова в названии." <<

endl;

        break;

    case 4:
        cout << "Введите id элемента, которого вы хотите получить: ";
        cin >> value;
        cout << endl;

        result = list.getProgramID(value);
        if (result == -5)
        {
            cout << "Элемент с таким ID отсутствует." << endl;
            break;
        }

        list.getOneEl(result);
        break;

    case 5:
        break;

    default:
        cout << "Неверный символ. Повторите попытку." << endl;
        break;
}
break;
case 2:
    cout << "Выберите команду:" << endl;
    cout << "1) Сохранить данные в файл" << endl;
    cout << "2) Читать данные из файла" << endl;
    cout << "3) Вернуться к выбору" << endl;
    cout << "===== " << endl;
    cout << "Ваш выбор: ";
    cin >> choise2;
    cout << endl;

    switch (choise2)
    {
    case 1:
        cout << "Введите название файла для записи данных программ: ";
        cin >> fileName;
        cout << endl;

        n = fileName.find(".txt");
        if (n > 187) fileName += string(".txt");

        list.saveFile(fileName);
        break;

    case 2:
        cout << "Введите название файла для чтения данных программ: ";
        cin >> fileName;
        cout << endl;

```

```

        n = fileName.find(".txt");
        if (n > 187) fileName += string(".txt");

        list.readFile(fileName);
        break;

    case 3:
        break;

    default:
        cout << "Неверный символ. Повторите попытку." << endl;
        break;
    }
    break;
case 3:
    cout << "Сортировать количество строк по:" << endl;
    cout << "1) Возрастанию" << endl;
    cout << "2) Убыванию" << endl;
    cout << "Ваш выбор: ";
    cin >> value;
    cout << endl;

    if (value == 1){
        list.sort(list.sortAsc);
    } else if (value == 2)
    { list.sort(list.sortDesc);
    } else cout << "Ошибка. Неверный номер команды." << endl;

    break;

case 4:
    cout << "Введите ID элемента, который хотите удалить: ";
    cin >> value;
    cout << endl;

    result = list.getProgramID(value);
    list.delProgram(result);

    break;

case 5:
    cout << "Выберите команду:" << endl;
    cout << "1) Добавить стандартную программу" << endl;
    cout << "2) Ввести данные с клавиатуры" << endl;
    cout << "3) Вернуться к выбору" << endl;
    cout << "=====" << endl;
    cout << "Ваш выбор: ";
    cin >> choice2;
    cout << endl;

    switch (choice2)
    {
    case 1:
        cout << "1) Добавить программу вредоносного ПО" << endl;
        cout << "2) Добавить программу на продажу" << endl;
        cout << "=====" << endl;
        cout << "Ваш выбор: ";
        cin >> choice3;
        cout << endl;

        if (choice3 == 1)
        {
            CProgram* newProgram = list.newProgram(listAuthors, 1);
            list.addProgram(newProgram);
            cout << "Программа добавлена." << endl;
            break;
        }
        else if (choice3 == 2)
        {

```



```

        CProgram* newProgram = list.newProgram(listAuthors, 2);
        list.addProgram(newProgram);
        cout << "Программа добавлена." << endl;
    }
    else
    {
        cout << "Неверный номер выбора." << endl;
    }

    break;

    case 2:
        list.enterNewProgram();
        break;

    case 3:
        break;

    default:
        cout << "Неверный символ. Повторите попытку." << endl;
        break;
    }

    break;

    case 6:
        cout << "Завершение работы." << endl;
        stop = 0;
        break;

    default:
        cout << "Неверный символ. Повторите попытку." << endl;
        break;
    }
}

author.deleteList();
return;
}

```

programList.h

```

#pragma once
#include "programForSale.h"
#include "malware.h"

class CList
{ private:
    int listSize;
    CProgram** programList;

public:
    int getListSize() const;

    void createlist(int, CAuthor*);
    void addProgram(CProgram*);
    void delProgram(int);
    CProgram* newProgram(CAuthor*, int) const;
    void printAll() const;
    sint task(int) const;
    int linesInFile(string) const;
    void readFile(string);
    void saveFile(string) const;
    void enterNewProgram();
    int regexTask() const;
    void getOneEl(int) const;

```

```

    int getProgramID(int) const;
    void sort(comp) const;
    static bool sortAsc(const int&, const int&);
    static bool sortDesc(const int&, const int&);

    int testID() const;
    stringstream testStringstream(int);
    CProgram* copyTest();

    CProgram* operator[] (int);

    ~CList();
};

```

programForSale.h

```

#pragma once
#include "program.h"
class CProgramForSale final : public CProgram
{
private:
    int price;

public:
    string getInfo() const override final;
    stringstream getStr() const override final;
    void input(istream&) override final;

    CProgramForSale();
    CProgramForSale(bool, int, int, int, int, string, CAuthor, sint, sint, sint, int);
    CProgramForSale(const CProgramForSale&);
    ~CProgramForSale() override final;

    CProgramForSale& operator= (CProgramForSale&);
    bool operator==(const string) const override final;
};

```

program.h

```

#pragma once
#include "author.h"
#include "date.h"
#define _CRT_SECURE_NO_WARNINGS
#define CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, FILE, __LINE)
#include <string>
#include <iostream>
#include <iomanip>
#include <locale>
#include <fstream>
#include <sstream>
#include <regex>

using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::setw;
using std::boolalpha;
using std::setiosflags;
using std::ios;
using std::ifstream;
using std::ofstream;
using std::stringstream;

```

```

using std::regex;
using std::regex_match;
using std::regex_search;
using std::cmatch;

typedef bool (comp)(const int&, const int&);

class CProgram {
protected:
    int timeOfWork;           //average time of program execution
    int size;                 //size of program
    int amountOfLines;        //number of lines in code
    int index;                //index
    bool useInternet;         //use internet
    string name;              //name of program
    CAuthor author;           //creator of program
    CDate date;               //date of creating program

public:
    int getTime() const;
    int getSize() const;
    int getLines() const;
    int getIndex()const;
    bool getInternet()const;
    string getName() const;

    virtual string getInfo() const = 0;
    virtual void input(istream&) = 0;
    virtual stringstream getStr() const = 0;

    CProgram();
    CProgram(bool, int, int, int, int, string, CAuthor, sint, sint, sint);
    CProgram(const CProgram& other);
    virtual ~CProgram();

    friend ostream& operator<< (ostream&,const CProgram&);
    friend ostream& operator<< (ostream&, const CProgram&);
    friend istream& operator>> (istream&, CProgram&);
    virtual bool operator==(const string) const;
    CProgram& operator= (CProgram&);
};

```

malware.h

```

#pragma once
#include "program.h"
class CMalware final: public CProgram
{
private:
    string type;

public:
    string getInfo() const override final;
    stringstream getStr() const override final;
    void input(istream&) override final;

    CMalware();
    CMalware(bool, int, int, int, int, string, CAuthor, sint, sint, sint, string);
    CMalware(const CMalware&);
    ~CMalware() override final;

    CMalware& operator= (CMalware&);
    bool operator==(const string) const override final;
};

```

date.h

```
#pragma once
#include <iostream>

using std::istream;
using std::ostream;
typedef short sint;

class CDate
{
private:
    sint day;
    sint month;
    sint year;

public:
    sint getDay() const;
    sint getMonth() const;
    sint getYear() const;

    CDate();
    CDate(sint, sint, sint);
    CDate(const CDate& other);
    ~CDate();

    friend istream& operator>> (istream&, CDate&);
    friend ostream& operator<< (ostream&, const CDate&);
};
```

author.h

```
#pragma once
#include <iostream>
#include <string>

using std::string;
using std::ostream;
using std::istream;

class CAuthor
{
private:
    string companyName;
    int listSize;
    CAuthor* list;

public:
    CAuthor* createList(int size);
    void deletelist();
    string getAuthor();

    CAuthor authors(int value);

    CAuthor();
    CAuthor(string author);
    CAuthor(const CAuthor& other);
    ~CAuthor();

    friend istream& operator>> (istream&, CAuthor&);
    friend ostream& operator<< (ostream&, const CAuthor&);
    bool operator==(const string) const;
};
```

programList.cpp

```
#include "programList.h"

int CList::getListSize() const
{
    return listSize;
}

void CList::createList(int value, CAuthor* authors)
{
    listSize = value;
    programList = new CProgram * [value];

    for (size_t i = 0; i < value; i++)
    {
        if (i == 0)
        {
            *(programList + i) = new CMalware();
        }
        else if (i == 1)
        {
            *(programList + i) = new CProgramForSale(true, 121, 222, 532, 1234, "Skype",
*(authors + 1), 2, 12, 2002, 3000);
        }
        else if (i == 2)
        {
            *(programList + i) = new CMalware(true, 1445, 532, 745, 7896, "Spider",
*(authors + 2), 6, 9, 1995, "Keylogger");
        }
        else if (i == 3)
        {
            *(programList + i) = new CProgramForSale(false, 333, 444, 555, 6745,
"Calculator", *(authors + 3), 1, 11, 2001, 200);
        }
        if (i == 4)
        {
            *(programList + i) = new CProgramForSale();
        }
    }
}

void CList::addProgram(CProgram* program)
{
    CProgram** newList = new CProgram * [listSize + 1];
    CProgram** temp = programList;

    for (size_t i = 0; i < listSize; i++)
    {
        *(newList + i) = *(programList + i);
    }
    newList[listSize++] = program;
    programList = newList;
    newList = temp;

    delete newList;
}

void CList::delProgram(int value)
{
    if (listSize == 0)
    {
        cout << "список программ пуст. возвращение с выбору действий." << endl;
        return;
    }
    if (value <= 0 || value > listSize)
    {
        cout << "ошибка. неверный номер элемента. возвращение." << endl;
        return;
    }
}
```

```

CProgram** newList = new CProgram * [listSize - 1];
CProgram** temp = programList;

for (size_t i = 0; i < value; i++)
    *(newList + i) = *(programList + i);
for (size_t i = value, j = value + 1; j < listSize; i++, j++)
    *(newList + i) = *(programList + j);

programList = newList;
newList = temp;
listSize--;
delete* (newList + value);
delete newList;
}
CProgram* CList::newProgram(CAuthor* list, int value) const
{
    if (value == 1)
    {
        CProgram* Program = new CMalware(true, 444, 555, 666, 6734, "Stealer", *(list + 4),
30, 11, 1967, "Trojan");
        return Program;
    }
    else
    {
        CProgram* Program = new CProgramForSale(false, 34, 69, 534, 5378, "Randomizer", *(list
+ 5), 15, 15, 2015, 1000000);
        return Program;
    }
}
void CList::printAll() const
{
    auto i = 0;
    if (listSize == 0)
    {
        cout << "Список пуст." << endl << endl;
        return;
    }
    else if (listSize < i || i < 0)
    {
        cout << "Неверный номер элемента." << endl << endl;
    }

    cout << endl << setiosflags(ios::left);
    cout << setw(12) << "   Время" << setw(12) << "Размер";
    cout << setw(10) << "Строки" << setw(11) << "Интернет";
    cout << setw(14) << "Индекс" << setw(21) << "Название";
    cout << setw(16) << "Год выпуска" << setw(14) << "Автор";
    cout << setw(14) << "Тип/Цена" << endl;

    for (; i < listSize; i++)
    {
        cout << setiosflags(ios::left) << setw(2) << i + 1 << " ";
        cout << *programList[i];
        cout << endl;
    }
    cout << endl;
}
void CList::getOneEl(int value) const
{
    stringstream temp = programList[value]->getStr();

    int time, size, lines, index;
    sint day, month, year;
    string name, name2;
    string internet2;
    bool internet;
    string author;
    string data;

```

```

temp >> time;
temp >> size;
temp >> lines;
temp >> boolalpha >> internet2;
temp >> index;
temp >> year;
temp >> author;
temp >> data;
temp >> name;
temp >> name2;

if (internet2 == "1") internet = true;
else internet = false;

if (name2 == "") name = name + " ";
else(name = name + " " + name2);

cout << endl << setiosflags(ios::left);
cout << setw(12) << "   Время" << setw(12) << "Размер";
cout << setw(10) << "Строки" << setw(11) << "Интернет";
cout << setw(14) << "Индекс" << setw(21) << "Название";
cout << setw(16) << "Год выпуска" << setw(15) << "Автор";
cout << setw(13) << "Тип/Цена" << endl << "   ";

cout << setw(10) << time;
cout << setw(12) << size;
cout << setw(10) << lines;
cout << setw(10) << internet;
cout << setw(14) << index;
cout << setw(25) << name;
cout << setw(12) << year;
cout << setw(15) << author;
cout << setw(14) << data;
cout << endl;
}

sint CList::task(int minimalSize) const
{
    int size = 0;

    for (size_t i = 0; i < listSize; i++)
    {
        if (programList[i]->getSize() > minimalSize && programList[i]->getName() == "Notepad")
        {
            cout << *programList[i];
            size++;
        }
    }

    if (size == 0)
    {
        cout << "Программы с такими параметрами отсутствуют." << endl;
    }

    return size;
}

void CList::saveFile(string filename) const
{
    std::ofstream fout(filename);
    if (!fout.is_open())
    {
        cout << "Невозможно открыть файл." << endl;
    }

    fout.setf(ios::left);
    fout << setw(12) << "   Время" << setw(12) << "Размер";
    fout << setw(13) << "Строки" << setw(11) << "Интернет";
    fout << setw(15) << "Индекс" << setw(24) << "Название";
    fout << setw(16) << "Год выпуска" << setw(14) << "Автор";
    fout << setw(12) << "Тип/Цена" << endl;
}

```

```

for (size_t i = 0; i < listSize; i++)
{
    fout << setiosflags(ios::left) << setw(2) << i + 1 << " ";
    fout << *programList[i];
    fout << endl;
}

cout << "Запись в файл завершена." << endl;
fout.close();
}

int CList::linesInFile(string filename) const
{
    int size = 0;
    string line;
    regex expressionMalware("([\\d]* [\\d]* [a-zA-ZА-Яа-я]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");
    regex expressionProgramForSelling("([\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");

    ifstream fin(filename);
    if (!fin.is_open())
    {
        cout << "Невозможно открыть файл. Возвращение в меню." << endl;
        return 0;
    }

    while (getline(fin, line))
    {
        if (regex_match(line, expressionMalware) || regex_match(line, expressionProgramForSelling))
            size++;
        else cout << "Строка в файле не прошла проверку." << endl;
    }

    fin.close();
    return size;
}

void CList::readFile(string filename)
{
    regex expressionMalware("([\\d]* [\\d]* [a-zA-ZА-Яа-я]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");
    regex expressionProgramForSelling("([\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");
    string data, line;
    int size = CList::linesInFile(filename);
    int i = 0, j = 0, a = 0;
    bool b;
    ifstream fin(filename);
    if (!fin.is_open())
    {
        cout << "Неверное название файла. Повторите попытку." << endl;
        return;
    }

    for (size_t i = 0; i < size; i++)
    {
        delete* (programList + i);
    }
    delete[] programList;
    programList = new CProgram * [size];

    while (getline(fin, line) && j < size)
    {
        b = 1;
        if (regex_match(line.c_str(), expressionMalware))
        {
            do {
                b = 0;

```



```

        a = line.find("--");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(" ");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(",");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(":");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(";");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find("_");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }
    } while (b == 1);

    std::istringstream temp(line);
    programList[i] = new CMalware;
    temp >> *programList[i];
    i++;
}
else if (regex_match(line.c_str(), expressionProgramForSelling))
{
    do {
        b = 0;

        a = line.find("--");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(" ");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }
    }
}

```

```

        a = line.find(",");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find("::");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find(";");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

        a = line.find("_");
        if (a != -1)
        {
            line.erase(a, 1);
            b = 1;
        }

    } while (b == 1);

    std::istringstream temp(line);

    programList[i] = new CProgramForSale;
    temp >> *programList[i];
    i++;
}

}

listSize = size;
fin.close();
cout << endl << "Чтение из файла завершено." << endl;
}

void CList::enterNewProgram()
{
    string data;
    regex expressionMalware("([\\d]* [\\d]* [a-zA-ZА-Яа-я]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");
    regex expressionProgramForSelling("([\\d]* [\\d]* [\\d]* [\\d]* [1|0]* [\\w]* [\\d]* [A-ZА-Я]+[\\wА-Яа-я,.;:-]* [\\wА-Яа-я,.;:-]*)");

    cout << "Введите данные в линию в таком порядке:" << endl;
    cout << "Индекс Время Тип/Цена Размер Строки Интернет(1/0) Компания Год Название" << endl;

    cin.ignore();
    getline(cin, data);

    data = data + " ";

    if (regex_match(data, expressionMalware))
    {
        std::istringstream temp(data);
        CProgram* newProgram = new CMalware;
        temp >> *newProgram;
        cout << endl;
        addProgram(newProgram);
    }
    else if (regex_match(data, expressionProgramForSelling))
    {

```

```

        std::istringstream temp(data);
        CProgram* newProgram = new CProgramForSale;
        temp >> *newProgram;
        cout << endl;
        addProgram(newProgram);
    }
    else
    {
        cout << endl << "Было введено неправильное имя." << endl;
        cout << "Завершение работы функции. " << endl;
    }
}

int CList::regexTask() const
{
    int value = 0;
    regex regular("(^[A-ZА-Я]+[\\wА-Яа-я.,;:-]* [\\wА-Яа-я.,;:-]+)");

    for (size_t i = 0; i < listSize; i++)
    {
        if (regex_match(programList[i]->getName(), regular))
        {
            cout << *programList[i] << endl;
            value++;
        }
    }
    cout << endl;

    return value;
}

int CList::getProgramID(int id) const
{
    int size = 0;

    for (size_t i = 0; i < listSize; i++)
        if (programList[i]->getIndex() == id)
        {
            return i;
        }

    if (size == 0)
    {
        return size = -5;
        cout << "\nПрограммы с таким ID нету.\n" << endl;
    }
}

void CList::sort(comp condition) const
{
    CProgram* temp;
    bool pr;

    do {
        pr = 0;
        for (size_t i = 0; i < listSize - 1; i++)
        {
            if (condition(programList[i]->getLines(), programList[i + 1]->getLines()))
            {
                temp = *(programList + i);
                *(programList + i) = *(programList + i + 1);
                *(programList + i + 1) = temp;
                pr = 1;
            }
        }
    } while (pr);
}

bool CList::sortAsc(const int& a, const int& b) { return a > b; }
bool CList::sortDesc(const int& a, const int& b) { return a < b; }

int CList::testID() const
{

```

```

        return programList[0]->getIndex();
    }
    stringstream CList::testStringstream(int value)
    {
        stringstream temp = programList[value]->getStr();
        return temp;
    }
    CProgram* CList::copyTest()
    {
        CProgram* newProgram = programList[0];
        return newProgram;
    }

    CList::~CList()
    {
        for (size_t i = 0; i < listSize; i++)
        {
            delete programList[i];
        }

        delete[] programList;
    }

    CProgram* CList::operator[] (int i)
    {
        if (listSize > i) return programList[i];
    }

```

programForSale.cpp

```

#include "programForSale.h"

stringstream CProgramForSale::getStr() const
{
    stringstream temp;

    temp << timeOfWork << " " << size << " " << amountOfLines << " "
        << useInternet << " " << index << " " << name << " "
        << date.getDay() << " " << date.getMonth() << " "
        << date.getYear() << " " << author << " " << price;

    return temp;
}

string CProgramForSale::getInfo() const
{
    stringstream temp;
    temp.setf(ios::left);

    temp << setw(10) << timeOfWork << setw(12) << size
        << setw(9) << amountOfLines << setw(12) << boolalpha << useInternet
        << setw(11) << index << setw(26) << name
        << setw(14) << date.getYear() << setw(12) << author
        << setw(14) << price;

    return temp.str();
}

void CProgramForSale::input(istream& input)
{
    input >> index >> timeOfWork >> price >> size >> amountOfLines >> useInternet >> author >>
    date >> name;
}

CProgramForSale::CProgramForSale(bool internet, int time, int size, int lines, int index, string
name, CAuthor author, sint day, sint month, sint year, int price) : CProgram(internet, time, size,
lines, index, name, author, day, month, year), price(price) {}

```

```

CProgramForSale::CProgramForSale() : CProgram(), price(2000) {}
CProgramForSale::CProgramForSale(const CProgramForSale& other) : CProgram(other), price(other.price)
{}
CProgramForSale::~CProgramForSale() {}

CProgramForSale& CProgramForSale::operator=(CProgramForSale& temp)
{
    if (this == &temp) return *this;

    CProgram::operator=(temp);
    int price = temp.price;

    return *this;
}
bool CProgramForSale::operator==(const string name) const
{
    return this->name == name;
}

```

program.cpp

```

#include "program.h"

int CProgram::getTime() const
{
    return timeOfWork;
}
int CProgram::getSize() const
{
    return size;
}
int CProgram::getLines() const
{
    return amountOfLines;
}
int CProgram::getIndex() const
{
    return index;
}
bool CProgram::getInternet()const
{
    return useInternet;
}
string CProgram::getName()const
{
    return name;
}

CProgram::CProgram(bool internet, int time, int size, int lines, int index, string name, CAuthor
author, sint day, sint month, sint year): useInternet(internet), timeOfWork(time), size(size),
amountOfLines(lines), index(index), name(name), author(author), date(day, month, year)
{
    //cout << "\nВызвался конструктор с параметрами";
}
CProgram::CProgram() : useInternet(false), timeOfWork(0), size(0), amountOfLines(0), index(0101),
name("Basic")
{
    //cout << "\nВызвался конструктор по умолчанию.";
}
CProgram::CProgram(const CProgram& other) : useInternet(other.useInternet),
timeOfWork(other.timeOfWork), size(other.size), amountOfLines(other.amountOfLines),
index(other.index), name(other.name), author(other.author), date(other.date)
{
    //cout << "\nВызвался конструктор копирования.";
}

```

```

CProgram::~CProgram()
{
    //cout << "\nВызвался деструктор";
}

ofstream& operator<< (ofstream& output, const CProgram& program)
{
    output << program.getInfo();
    return output;
}

ostream& operator<< (ostream& output, const CProgram& program)
{
    output << program.getInfo();
    return output;
}

istream& operator>> (istream& input, CProgram& program)
{
    program.input(input);
    return input;
}

bool CProgram::operator==(const string name) const
{
    return this->name == name;
}

CProgram& CProgram::operator= (CProgram& temp)
{
    if (this == &temp) return *this;

    int timeOfWork = temp.timeOfWork;
    int size = temp.size;
    int amountOfLines = temp.amountOfLines;
    int index = temp.index;
    bool useInternet = temp.useInternet;
    string name = temp.name;
    CAuthor author = temp.author;
    CDate date = temp.date;

    return *this;
}

```

malware.cpp

```

#include "malware.h"
stringstream CMalware::getStr() const
{
    stringstream temp;

    temp << timeOfWork << " " << size << " " << amountOfLines << " "
        << useInternet << " " << index << " " << date.getYear() << " "
        << author << " " << type << " " << name << " ";

    return temp;
}

string CMalware::getInfo() const
{
    stringstream temp;
    temp.setf(ios::left);

    temp << setw(10) << timeOfWork << setw(12) << size
        << setw(9) << amountOfLines << setw(12) << boolalpha << useInternet
        << setw(11) << index << setw(26) << name
        << setw(14) << date.getYear() << setw(12) << author
        << setw(14) << type;

    return temp.str();
}

void CMalware::input(istream& input)
{

```

```

        input >> index >> timeOfWork >> type >> size >> amountOfLines >> useInternet >> author >>
date >> name;
}

CMalware::CMalware(bool internet, int time, int size, int lines, int index, string name, CAuthor
author, sint day, sint month, sint year, string type) : CProgram(internet, time, size, lines, index,
name, author, day, month, year), type(type) {}
CMalware::CMalware() : CProgram(), type("Exploit") {}
CMalware::CMalware(const CMalware& other) : CProgram(other), type(other.type) {}
CMalware::~CMalware() {}

CMalware& CMalware::operator=(CMalware& temp)
{
    if (this == &temp) return *this;

    CProgram::operator=(temp);
    string type = temp.type;

    return *this;
}
bool CMalware::operator==(const string name) const
{
    return this->name == name;
}

```

date.cpp

```

#include "date.h"

sint CDate::getDay() const
{
    return day;
}
sint CDate::getMonth() const
{
    return month;
}
sint CDate::getYear() const
{
    return year;
}

CDate::CDate() : day(1), month(1), year(1999) {}
CDate::CDate(sint day, sint month, sint year) : day(day), month(month), year(year) {}
CDate::CDate(const CDate& other) : day(other.day), month(other.month), year(other.year) {}
CDate::~CDate() {}

istream& operator>> (istream& input, CDate& date)
{
    input >> date.year;

    return input;
}
ostream& operator<< (ostream& output, const CDate& date)
{
    output << date.year;

    return output;
}

```

author.cpp

```
#include "author.h"

CAuthor* CAuthor::createList(int size)
{
    list = new CAuthor[size];
    listSize = size;

    for (size_t i = 0; i < size; i++)
    {
        list[i] = authors(i);
    }

    return list;
}

void CAuthor::deleteList()
{
    delete[] list;
}

string CAuthor::getAuthor()
{
    return companyName;
}

CAuthor CAuthor::authors(int value)
{
    if (value == 0)
    {
        CAuthor author("Oracle");
        return author;
    }
    else if (value == 1)
    {
        CAuthor author2("Lambda");
        return author2;
    }
    else if (value == 2)
    {
        CAuthor author3("AMD");
        return author3;
    }
    else if (value == 3)
    {
        CAuthor author4("Logitech");
        return author4;
    }
    else if (value == 4)
    {
        CAuthor author5("Hynix");
        return author5;
    }
    else if (value == 5)
    {
        CAuthor author6("Mojang");
        return author6;
    }
}

CAuthor::CAuthor() : companyName("IBM") {}
CAuthor::CAuthor(string author) : companyName(author) {}
CAuthor::CAuthor(const CAuthor& other) : companyName(other.companyName) {}
CAuthor::~CAuthor() {}

istream& operator>> (istream& input, CAuthor& author)
{
    input >> author.companyName;
    return input;
}
```



```
ostream& operator<< (ostream& output, const CAuthor& author)
{
    output << author.companyName;
    return output;
}
bool CAuthor::operator==(const string author) const
{
    return this->companyName == author;
}
```

4. Результати роботи програми

```

Выберите команду для работы со списком:
1)Вывод на экран
2)Работа с файлами
3)Сортировка данных
4)Удаление элемента
5)Добавление элементов
6)Завершение работы
=====
Ваш выбор: 1

Выберите команду:
1) Вывести весь список на экран
2) Вывести список программ больше определённого размера за авторством Microsoft
3) Вывести список программ, названия которых состоят из 2 слов
4) Вывести программу по ID
5) Вернуться к выбору действий
=====
Ваш выбор: 1


```

	Время	Размер	Строки	Интернет	Индекс	Название	Год выпуска	Автор	Тип/Цена
1)0	0	0	0	false	65	Basic	1999	IBM	Exploit
2)121	222	532	true	1234	Skype	2002	Lambda	3000	
3)1445	532	745	true	7896	Spider	1995	AMD	Keylogger	
4)333	444	555	false	6745	Calculator	2001	Logitech	200	

```

Тест получения элемента по ID                выполнен успешно.
Элемент добавлен.
Тест добавления элемента в список             выполнен успешно.
Тест функции удаления                         выполнен успешно.

Программы с такими параметрами отсутствуют.

Тест нахождения элементов по параметру         выполнен успешно.
Тест функции stringstream                     выполнен успешно.
Строка в файле не прошла проверку.
Тест функции чтения из файла                  выполнен успешно.

Тест функции regex                            выполнен успешно.
Тест функции сортировки                       выполнен успешно.
Тест агрегации                               выполнен успешно.

Утечка памяти отсутствует.

```

5. Висновки

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з перевантаженням методів.

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