Звіт

Автор: Чугунов В.Ю

КІТ-119а

Лабораторна робота №8

**ПЕРЕВАНТАЖЕННЯ ОПЕРАТОРІВ. СЕРІАЛІЗАЦІЯ**

**Мета:** Отримати знання про призначення операторів, визначити їх ролі в житті об’єкта та можливість перевантаження.

1 ЗАВДАННЯ ДО РОБОТИ

**Загальне завдання.** Поширити попередню роботу наступним чином:  
 - в базовому класі, та класах-спадкоємцях перевантажити:  
 - оператор присвоювання;  
 - оператор порівняння;  
 - оператор введення/виведення;  
 - в класі-списку перевантажити:  
 - оператор індексування;  
 - оператор введення/виведення.

2 РОЗРОБКА АЛГОРИТМІВ ФУНКЦІЙ

**Клас Lessons**  
 **private поля:**

string audienceID – номер аудиторії

string lecturerName – прізвище лектора

string lessonType – назва предмета

int numOfWorks – кількість самостійних робіт

int lessonStartTime – час початку занять

**Методи:**  
 **void operator= ( const Lessons & obj )**  
 Призначення: перевантаження оператора присвоювання.  
 Схема алгоритму подана на рис. 1.  
   
 **bool operator== ( const Lessons & obj )** Призначення: перевантаження оператора порівняння.  
 Схема алгоритму подана на рис. 2.  
  
 **ostream & operator<< ( ostream & os, const Lessons & obj )**  
 Призначення: перевантаження оператора виведення інформації.  
 Схема алгоритму подана на рис. 3.  
  
 **istream & operator>> ( istream & is, Lessons & obj )** Призначення: перевантаження оператора введення інформації.  
 Схема алгоритму подана на рис. 4.

**Клас Works  
 private поля:**

Node \*head – покажчик на перший вузол списку

int size – загальна кількість вузлів

int sumOfHWs – загальна кількість домашніх робіт

int sumOfWorks – загальна кількість самостійних робіт

**Методи:**  
 **Node \* operator[] ( cons tint pos )**  
 Призначення: перевантаження оператора індексування.  
 Схема алгоритму подана на рис. 5.  
  
 **ostream & operator<< ( ostream & os, Works & w )** Призначення: перевантаження оператора виведення інформації про усі елементи класу-списку.  
 Схема алгоритму подана на рис. 6.  
  
 **ostream & operator<< ( ostream & os, Works::Node \* w )**  
 Призначення: перевантаження оператора виведення інформації про елемент, отриманий по індексу.  
 Схема алгоритму подана на рис. 7.

**Схеми алгоритмів функцій**

Рисунок 1 – Схема алгоритму оператора « = » класу Lessons

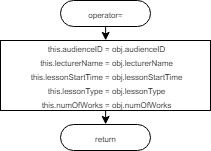


Рисунок 2 – Схема алгоритму оператора « == » класу Lessons

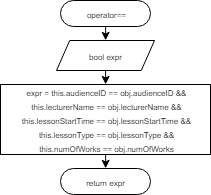


Рисунок 3 – Схема алгоритму оператора « << » класу Lessons

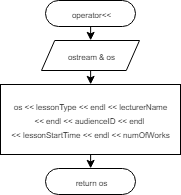


Рисунок 4 – Схема алгоритму оператора « >> » класу Lessons

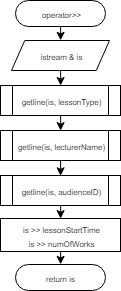


Рисунок 5 – Схема алгоритму оператора « [] » класу Works

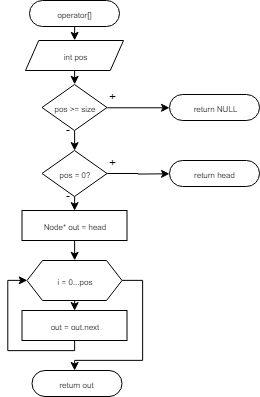
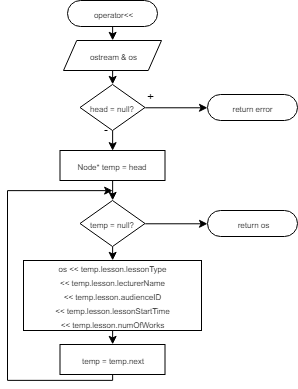


Рисунок 6 – Схема алгоритму оператора « << » класу Works



3 ТЕКСТ ПРОГРАМИ

**Текст файлу Lessons.h**

#pragma once

#include <iostream>

#include <string>

using std::string;

using std::cout;

using std::cin;

using std::endl;

using std::ostream;

using std::istream;

class Lessons

{

private:

string audienceID;

string lecturerName;

string lessonType;

int numOfWorks;

int lessonStartTime;

public:

Lessons();

Lessons(const string t, const string name, const string ID, const int time, const int qty);

Lessons(const Lessons & copyObject);

//Accessors

void set\_audienceID(const string index) { this->audienceID = index; };

void set\_lecturerName(const string name) { this->lecturerName = name; };

void set\_lessonType(const string type) { this->lessonType = type; };

void set\_numOfWorks(const int number) { this->numOfWorks = number; };

void set\_lessonStartTime(const int time) { this->lessonStartTime = time; };

inline const string& get\_audienceID() const { return this->audienceID; };

inline const string& get\_lecturerName() const { return this->lecturerName; };

inline const string& get\_lessonType() const { return this->lessonType; };

inline const int& get\_numOfWorks() const { return this->numOfWorks; };

inline const int& get\_lessonStartTime() const { return this->lessonStartTime; };

//Functions

virtual void generate();

//Operators

void operator= (const Lessons &obj);

bool operator== (const Lessons &obj);

friend ostream& operator<< (ostream& os, const Lessons& obj);

friend istream& operator>> (istream& is, Lessons& obj);

};

ostream& operator<< (ostream& os, const Lessons& obj);

istream& operator>> (istream& is, Lessons& obj);

**Текст файлу Lessons.cpp**

#include <regex>

#include "Lessons.h"

using std::regex;

using std::regex\_match;

using std::getline;

Lessons::Lessons()

{

audienceID = "NULL";

lecturerName = "NULL";

lessonType = "NULL";

lessonStartTime = 0;

numOfWorks = 0;

}

Lessons::Lessons(const string t, const string name, const string ID, const int time, const int qty)

{

audienceID = ID;

lecturerName = name;

lessonType = t;

lessonStartTime = time;

numOfWorks = qty;

}

Lessons::Lessons(const Lessons & copyObject)

{

this->audienceID = copyObject.audienceID;

this->lecturerName = copyObject.lecturerName;

this->lessonStartTime = copyObject.lessonStartTime;

this->lessonType = copyObject.lessonType;

this->numOfWorks = copyObject.numOfWorks;

}

void Lessons::generate()

{

regex allowableStr("([A-Z])""(\\w)\*(\\s)?(\.)?");

cout << "\nEnter the subject name: ";

cin.ignore(20, '\n');

do {

getline(cin, lessonType);

if (!regex\_match(lessonType, allowableStr))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!regex\_match(lessonType, allowableStr));

cout << "Enter lecturer name: ";

do {

getline(cin, lecturerName);

if (!regex\_match(lecturerName, allowableStr))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!regex\_match(lecturerName, allowableStr));

cout << "Enter an audience ID: ";

do {

getline(cin, audienceID);

if (!(regex\_match(audienceID, allowableStr)))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!(regex\_match(audienceID, allowableStr)));

cout << "Enter lesson start time: ";

cin >> lessonStartTime;

cout << "Enter a number of works student has completed per semester: ";

cin >> numOfWorks;

}

void Lessons::operator=(const Lessons & obj)

{

this->set\_audienceID(obj.get\_audienceID());

this->set\_lecturerName(obj.get\_lessonType());

this->set\_lessonStartTime(obj.get\_lessonStartTime());

this->set\_lessonType(obj.get\_lessonType());

this->set\_numOfWorks(obj.get\_numOfWorks());

}

bool Lessons::operator==(const Lessons & obj)

{

bool expr = this->audienceID == obj.audienceID

&& this->lecturerName == obj.lecturerName

&& this->lessonType == obj.lessonType

&& this->numOfWorks == obj.numOfWorks

&& this->lessonStartTime == obj.lessonStartTime;

if (expr)

return true;

else

return false;

}

ostream & operator<<(ostream & os, const Lessons & obj)

{

os << "Subject: " << obj.get\_lessonType() << endl

<< "Lecturer: " << obj.get\_lecturerName() << endl

<< "Audience ID: " << obj.get\_audienceID() << endl

<< "Lesson start time: " << obj.get\_lessonStartTime() << endl

<< "The number of works student wrote for the semester: " << obj.get\_numOfWorks() << endl;

return os;

}

istream & operator>>(istream & is, Lessons & obj)

{

regex allowableStr("([A-Z])""(\\w)\*(\\s)?(\.)?");

cout << "\nEnter the subject name: ";

do {

getline(is, obj.lessonType);

if (!regex\_match(obj.lessonType, allowableStr))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!regex\_match(obj.lessonType, allowableStr));

cout << "Enter lecturer name: ";

do {

getline(is, obj.lecturerName);

if (!regex\_match(obj.lecturerName, allowableStr))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!regex\_match(obj.lecturerName, allowableStr));

cout << "Enter an audience ID: ";

do {

getline(is, obj.audienceID);

if (!(regex\_match(obj.audienceID, allowableStr)))

cout << "Error (invalid syntax)!\nRepeat entering: ";

} while (!(regex\_match(obj.audienceID, allowableStr)));

cout << "Enter lesson start time: ";

is >> obj.lessonStartTime;

cout << "Enter a number of works student has completed per semester: ";

is >> obj.numOfWorks;

return is;

}

**Текст файлу Works.h**

#pragma once

#include "Lessons.h"

#include "Home.h"

class Works

{

public:

Works();

~Works();

//Functions

void push\_back(Lessons\* lesson);

void pop\_front();

void erase(int pos);

void show();

void clear();

//Accesssors

inline const int& get\_works() const { return this->sumOfWorks; };

inline const int& get\_homeworks() const { return this->sumOfHWs; };

inline const int& get\_size() const { return this->size; };

//Inner classes

class Node

{

public:

Lessons \*lesson;

Node \*next;

Node(Lessons \*l = nullptr, Node \*n = nullptr);

};

//Operators

Node\* operator[] (const int pos);

friend ostream& operator<< (ostream& os, Works& w);

friend ostream& operator<< (ostream& os, Works::Node\* w);

private:

Node \*head;

int size;

int sumOfHWs;

int sumOfWorks;

};

ostream& operator<< (ostream& os, Works& w);

ostream& operator<< (ostream& os, Works::Node\* w);

**Текст файлу Works.cpp**

#include <iostream>

#include "Works.h"

Works::Works()

{

head = nullptr;

size = 0;

sumOfHWs = 0;

sumOfWorks = 0;

}

Works::~Works()

{

clear();

}

void Works::push\_back(Lessons\* lesson)

{

if (!size)

{

head = new Node(lesson);

sumOfWorks += lesson->get\_numOfWorks();

if (((Home \*)lesson)->get\_numOfHWs() > 0)

sumOfHWs += ((Home \*)lesson)->get\_numOfHWs();

}

else

{

Node \*current = this->head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = new Node(lesson);

sumOfWorks += lesson->get\_numOfWorks();

if (((Home \*)lesson)->get\_numOfHWs() > 0)

sumOfHWs += ((Home \*)lesson)->get\_numOfHWs();

}

this->size++;

}

void Works::erase(int pos)

{

if (!pos)

pop\_front();

else

{

Node \*previous = head;

for(int i = 0; i < pos-1; i++)

previous = previous->next;

Node \*temp = previous->next;

previous->next = temp->next;

delete temp;

temp = nullptr;

size--;

}

}

void Works::show()

{

if (this->head == nullptr)

cout << "\nError (list is empty)\n";

else

{

Node \*temp = head;

int i = 0;

while (temp)

{

cout << "\nInfo about lesson #" << ++i << ": " << endl;

cout << "Subject: " << temp->lesson->get\_lessonType() << endl;

cout << "Lecturer: " << temp->lesson->get\_lecturerName() << endl;

cout << "Audience ID: " << temp->lesson->get\_audienceID() << endl;

cout << "Lesson start time: " << temp->lesson->get\_lessonStartTime() << endl;

cout << "The number of works student wrote for the semester: " << temp->lesson->get\_numOfWorks() << endl;

if (((Home \*)temp->lesson)->get\_numOfHWs() > 0)

cout << "The number of hometasks student wrote per semester: " << ((Home \*)temp->lesson)->get\_numOfHWs() << endl;

temp = temp->next;

}

}

}

void Works::pop\_front()

{

Node \*temp = head;

head = head->next;

delete temp;

size--;

}

void Works::clear()

{

while (size)

{

pop\_front();

}

}

Works::Node\* Works::operator[](const int pos)

{

if (pos >= size)

{

cout << "Error (out of range)";

return nullptr;

}

if (!pos)

{

return head;

}

Node \*out = head;

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

out = out->next;

return out;

}

Works::Node::Node(Lessons \*l, Node \*n)

{

this->lesson = l;

this->next = n;

}

ostream& operator<<(ostream & os, Works& w)

{

if (!w.head)

{

os << "\nError (list is empty)\n";

return os;

}

Works::Node \*temp = w.head;

int i = 0;

while (temp)

{

os << "\nInfo about lesson #" << ++i << ": " << endl

<< "Subject: " << temp->lesson->get\_lessonType() << endl

<< "Lecturer: " << temp->lesson->get\_lecturerName() << endl

<< "Audience ID: " << temp->lesson->get\_audienceID() << endl

<< "Lesson start time: " << temp->lesson->get\_lessonStartTime() << endl

<< "The number of works student wrote for the semester: " << temp->lesson->get\_numOfWorks() << endl;

if (((Home \*)temp->lesson)->get\_numOfHWs() > 0)

os << "The number of hometasks student wrote per semester: " << ((Home \*)temp->lesson)->get\_numOfHWs() << endl;

temp = temp->next;

}

return os;

}

ostream & operator<<(ostream & os, Works::Node \* w)

{

if (!w->lesson)

{

os << "Error (Bad Index)";

return os;

}

os << "\nSubject: " << w->lesson->get\_lessonType() << endl

<< "Lecturer: " << w->lesson->get\_lecturerName() << endl

<< "Audience ID: " << w->lesson->get\_audienceID() << endl

<< "Lesson start time: " << w->lesson->get\_lessonStartTime() << endl

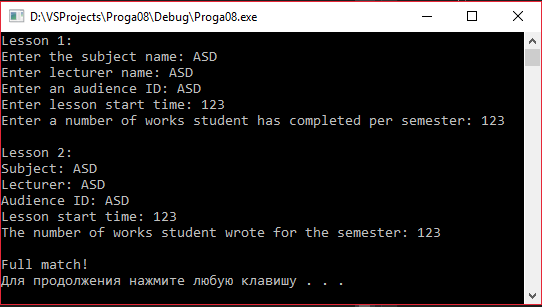
<< "The number of works student wrote for the semester: " << w->lesson->get\_numOfWorks() << endl;

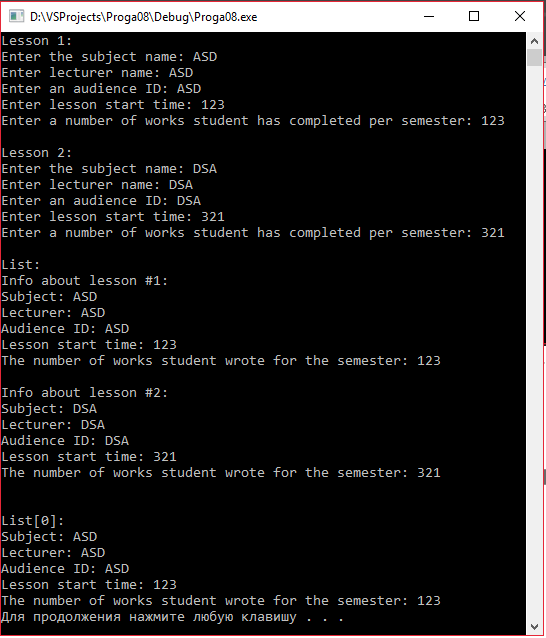
if (((Home \*)w->lesson)->get\_numOfHWs() > 0)

os << "The number of hometasks student wrote per semester: " << ((Home \*)w->lesson)->get\_numOfHWs() << endl;

return os;

}

4 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ  
   
Рисунок 8 – Результат роботи операторів « = », « << », « >> », « == »

  
Рисунок 9 – Результат роботи операторів « [] », « << »

ВИСНОВКИ

При виконанні лабораторної роботи було поширено попередню роботу (Класи. Поліморфізм) наступним чином:  
 - в базовому класі, та класах-спадкоємцях перевантажено:  
 - оператор присвоювання;  
 - оператор порівняння;  
 - оператор введення/виведення;  
 - в класі-списку перевантажено:  
 - оператор індексування;  
 - оператор введення/виведення.  
 Досліджено методи перевантаження операторів.