Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Об’єктно – орієнтоване програмування

ЗВІТ ДО

ЛАБОРАТОРНОЇ РОБОТИ №6

ВАРІАНТ №7

Оцінка «\_\_\_\_\_\_\_»

Дата «\_\_» \_\_\_\_\_\_\_ 2017р

Виконав студент ІІ курсу

Гр. ТІ-62

Заїчко Олексій Павлович

Перевірив:\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2017

***Склад програми:***

main.cpp

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <Windows.h>

#include "Studio.h"

#include "OperationList.h"

#include "CustomArray.h"

#include "FileError.h"

#include <vector>

#include <algorithm>

int getTypeSizeFromId(const std::type\_info& p);

bool durationCmp(const Operation<char\*>& a, const Operation<char\*>& b);

int main()

{

srand(time(NULL));

Operation<std::string> test(1, 2017);

Operation<std::string> test\_2( 5, 2015, "Leha", "Goreliy");

std::ofstream file;

file.open("test.bin", std::ios::binary);

if(file.is\_open())

{

CustomArray<Operation<std::string>> arr(2);

arr.insert(test);

arr.insert(test\_2);

arr.printBinary(file);

file.close();

}

else std::cout << "Unable to open file\n\n";

file.open("test.txt");

if (file.is\_open())

{

CustomArray<Operation<std::string>> arr(2);

arr.insert(test);

arr.insert(test\_2);

arr.printText(file);

file.close();

}

else std::cout << "Unable to open file\n\n";

/\* <char\*> and <std::string> sort time comparasion\*/

OperationList<unsigned int, std::string> listS(200, 2, 3, 2017);

OperationList<unsigned int, char\*> listC(200, 2, 3, 2017);

listC.fill();

listS.fill();

long start = GetTickCount();

listC.sortByName();

long end = GetTickCount();

std::cout << "Size of Operation<char\*>: " << sizeof(Operation<char\*>) << std::endl;

std::cout << "Average duration: " << getAverageDuration(listC) << std::endl;

std::cout << "Operation<char\*> sort time: " << end - start << std::endl << std::endl;

start = GetTickCount();

listS.sortByName();

end = GetTickCount();

std::cout << "Size of Operation<std::string>: " << sizeof(Operation<std::string>) << std::endl;

std::cout << "Average duration: " << getAverageDuration(listS) << std::endl;

std::cout << "Operation<std::string> sort time: " << end - start << "\n" << std::endl;

OperationList<unsigned int, std::string> copy = listS;

//std::cout << listC << listS << std::endl;

/\* Sort speed comparasion\*/

start = GetTickCount();

listC.sortByDuration();

end = GetTickCount();

std::cout << "Operation<char\*> selection sort by int field time: " << end - start << "\n" << std::endl;

start = GetTickCount();

listC.sortByName();

end = GetTickCount();

std::cout << "Operation<char\*> selection sort by char\* field time: " << end - start << "\n" << std::endl;

start = GetTickCount();

listC.sortByDuration();

end = GetTickCount();

std::cout << "Operation<char\*> selection sort by int field time using move operations: " << end - start << "\n" << std::endl;

start = GetTickCount();

listC.sortByName();

end = GetTickCount();

std::cout << "Operation<char\*> selection sort by char\* field time using move operations: " << end - start << "\n" << std::endl;

std::vector<Operation<char\*>> vector;

int quantity = listC.getQuantity();

for (int i = 0; i < quantity; i++) {

vector.push\_back(listC[i]);

}

start = GetTickCount();

std::sort(vector.begin(), vector.end(), durationCmp);

end = GetTickCount();

std::cout << "Operation<char\*> sort() by int field time: " << end - start << "\n" << std::endl;

start = GetTickCount();

std::sort(vector.begin(), vector.end());

end = GetTickCount();

std::cout << "Operation<char\*> sort() by char\* field time: " << end - start << "\n" << std::endl;

/\* Working w/ typeid() \*/

Date\_2 d2(12, 2012);

OperationDate opDate(1, 3, 2018);

Abonent<char\*> abChar(6, 4, 1999, "Leha", "Char");

Abonent<std::string> abString(6, 4, 1999, "Leha", "String");

Disc<char\*> discChar(1, 1, "Nazarchuk", "2 Chara", 20, 2, 4, 1985);

Disc<std::string> discString(1, 1, "Nazarchuk", "2 Stringa", 20, 2, 4, 1985);

std::ofstream outfile;

outfile.open("test.dat", std::ios::binary);

if (outfile.is\_open())

{

Date\_2\*\* arr = new Date\_2\*[6];

arr[0] = &d2;

arr[1] = &opDate;

arr[2] = &abChar;

arr[3] = &abString;

arr[4] = &discChar;

arr[5] = &discString;

int temp;

for (int i = 0; i < 6; i++) {

temp = getTypeSizeFromId(typeid(\*(arr[i])));

std::cout << temp << " " << typeid(\*arr[i]).name() << std::endl;

outfile.write(reinterpret\_cast<const char\*>(arr[i]), temp);

}

outfile.close();

}

std::ifstream infile;

infile.open("test.dat", std::ios::binary);

if (infile.is\_open())

{

Date\_2\*\* arr = new Date\_2\*[6];

arr[0] = new Date\_2;

arr[1] = new OperationDate;

arr[2] = new Abonent<char\*>;

arr[3] = new Abonent<std::string>;

arr[4] = new Disc<char\*>;

arr[5] = new Disc<std::string>;

int temp;

for (int i = 0; i < 6; i++) {

temp = getTypeSizeFromId(typeid(\*(arr[i])));

infile.read(reinterpret\_cast<char\*>(arr[i]), temp);

arr[i]->printConsole();

}

infile.close();

}

///////////////////////////////////////////

system("pause>nul");

return 0;

}

bool durationCmp(const Operation<char\*>& a, const Operation<char\*>& b) {

return a.getFilmDuration() < b.getFilmDuration();

}

int getTypeSizeFromId(const std::type\_info& p) {

if (p == typeid(Date\_2) || p == typeid(ProductionDate) || p == typeid(BookingDate)

|| p == typeid(OperationDate) || p == typeid(OperationListDate)) {

return sizeof(Date\_2);

}

if (p == typeid(Abonent<char\*>)) {

return sizeof(Abonent<char\*>);

}

if (p == typeid(Abonent<std::string>)) {

return sizeof(Abonent<std::string>);

}

if (p == typeid(Disc<char\*>)) {

return sizeof(Disc<char\*>);

}

if (p == typeid(Disc<std::string>)) {

return sizeof(Disc<std::string>);

}

}

char\* genString(const int len) {

int size = rand() % len + 1;

char\* s = new char[size + 1];

static const char alphanum[] =

"0123456789"

"ABCDEFGHIJKLMNOPQRSTUVWXYZ"

"abcdefghijklmnopqrstuvwxyz";

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

s[i] = alphanum[rand() % (sizeof(alphanum) - 1)];

}

s[size] = '\0';

return s;

}

unsigned int getUnsigned(unsigned int max, unsigned int min) {

return rand() % (max - min + 1) + min;

}

Date\_1.h

#include <iostream>

#include <iomanip>

#include <fstream>

class Date\_1 {

private:

int year;

short month;

public:

Date\_1(short month = 1, int year = 1980);

Date\_1(const Date\_1& p);

virtual void safeSetMonth(short mm);

void safeSetYear(int yy);

void setMonth(short mm);

void setYear(int yy);

short getMonth() const;

int getYear() const;

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

friend std::ostream& operator<<(std::ostream& os, const Date\_1& p);

};

class FoundationDate : public Date\_1 {

public:

FoundationDate(short mm, int yy):Date\_1(mm,yy){}

FoundationDate(const FoundationDate& p): Date\_1(p){}

friend std::ostream& operator<<(std::ostream& os, const FoundationDate& p);

};

Date\_1.cpp

#include "Date\_1.h"

Date\_1::Date\_1(short mm, int yy) {

month = (mm > 0 && mm < 13) ? mm : 1;

year = (yy > 1979 && yy < 10000) ? yy : 1980;

}

Date\_1::Date\_1(const Date\_1& p) {

month = p.month;

year = p.year;

}

void Date\_1::safeSetMonth(short mm) {

month = (mm > 0 && mm < 13) ? mm : month;

}

void Date\_1::safeSetYear(int yy) {

year = (yy >= 1980 && yy <= 9999) ? yy : year;

}

void Date\_1::setMonth(short mm) {

month = mm;

}

void Date\_1::setYear(int yy) {

year = yy;

}

short Date\_1::getMonth() const {

return month;

}

int Date\_1::getYear() const {

return year;

}

std::ostream& operator<<(std::ostream& os, const Date\_1& p) {

char temp = os.fill();

os << "Date\_1 data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Date\_1::printText(std::ofstream& os) {

char temp = os.fill();

os.fill(' ');

os.put(' ');

os << std::setw(2) << getMonth();

os.put(' ');

os << std::setw(4) << getYear();

os.put('\n');

}

void Date\_1::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

std::ostream& operator<<(std::ostream& os, const FoundationDate& p) {

char temp = os.fill();

os << "FoundationDate data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

Date\_2.h

#include <iostream>

#include <iomanip>

#include <fstream>

class Date\_2 {

private:

int year;

short month;

short day;

static int monthsDays[13];

public:

Date\_2(int day = 1, int month = 1, int year = 1980);

Date\_2(const Date\_2& p);

void safeSetDate(int dd = 1, int mm = 1, int yy = 1980);

void setDate(int dd = 1, int mm = 1, int yy = 1980);

void setDay(int dd);

void setMonth(int mm);

void setYear(int yy);

int getDay() const;

int getMonth() const;

int getYear() const;

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

virtual void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Date\_2& p);

};

class ProductionDate : public Date\_2 {

public:

ProductionDate(int day = 1, int month = 1, int year = 1980)

:Date\_2(day, month, year){}

ProductionDate(const ProductionDate& p): Date\_2(p){}

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const ProductionDate& p);

};

class BookingDate : public Date\_2 {

public:

BookingDate(int day = 1, int month = 1, int year = 1980)

:Date\_2(day, month, year) {}

BookingDate(const BookingDate& p) : Date\_2(p) {}

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const BookingDate& p);

};

class OperationDate : public Date\_2 {

public:

OperationDate(int day = 1, int month = 1, int year = 1980)

:Date\_2(day, month, year) {}

OperationDate(const OperationDate& p) : Date\_2(p) {}

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const OperationDate& p);

};

class OperationListDate : public Date\_2 {

public:

OperationListDate(int day = 1, int month = 1, int year = 1980)

:Date\_2(day, month, year) {}

OperationListDate(const OperationListDate& p) : Date\_2(p) {}

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const OperationListDate& p);

};

Date\_2.cpp

#include "Date\_2.h"

int Date\_2::monthsDays[13] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };

Date\_2::Date\_2(int day, int month, int year) {

safeSetDate(day, month, year);

}

Date\_2::Date\_2(const Date\_2& p) {

day = p.day;

month = p.month;

year = p.year;

}

void Date\_2::safeSetDate(int dd, int mm, int yy) {

month = (mm >= 1 && mm <= 12) ? mm : 1;

year = (yy >= 1980 && yy <= 3000) ? yy : 1980;

if (month == 2 && year % 4 == 0 && (year % 400 == 0 || year % 100 != 0)) {

day = (dd >= 1 && dd <= 29) ? dd : 1;

}

else {

day = (dd >= 1 && dd <= monthsDays[month]) ? dd : 1;

}

}

void Date\_2::setDate(int dd, int mm, int yy) {

day = dd;

month = mm;

year = yy;

}

void Date\_2::setDay(int dd) {

safeSetDate(dd, month, year);

}

void Date\_2::setMonth(int mm) {

safeSetDate(day, mm, year);

}

void Date\_2::setYear(int yy) {

safeSetDate(day, month, yy);

}

int Date\_2::getDay() const {

return day;

}

int Date\_2::getMonth() const {

return month;

}

int Date\_2::getYear() const {

return year;

}

std::ostream& operator<<(std::ostream& os, const Date\_2& p) {

char temp = os.fill();

os << "Date\_1 data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(3) << p.getDay();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

std::ostream& operator<<(std::ostream& os, const ProductionDate& p) {

char temp = os.fill();

os << "ProductionDate data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(3) << p.getDay();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

std::ostream& operator<<(std::ostream& os, const BookingDate& p) {

char temp = os.fill();

os << "BookingDate data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(3) << p.getDay();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

std::ostream& operator<<(std::ostream& os, const OperationDate& p) {

char temp = os.fill();

os << "OperationDate data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(3) << p.getDay();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

std::ostream& operator<<(std::ostream& os, const OperationListDate& p) {

char temp = os.fill();

os << "OperationListDate data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(3) << p.getDay();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Date\_2::printText(std::ofstream& os) {

os << std::setw(3) << getDay();

os.put(' ');

os << std::setw(5) << getMonth();

os.put(' ');

os << std::setw(4) << getYear();

os.put(' ');

os.put('\n');

}

void Date\_2::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

Abonent.h

#pragma once

#include "Date\_2.h"

#include <cstring>

#include <string>

template <class StrType>

class Abonent : public BookingDate {

};

template <>

class Abonent<char\*> : public BookingDate {

private:

char\* name;

char\* surname;

public:

Abonent(int dd = 1, int mm = 1, int yy = 1980,

char\* name = "Vasia", char\* surname = "Pupkin");

Abonent(const Abonent<char\*>& p);

Abonent(Abonent<char\*>&& p);

~Abonent();

const Abonent<char\*>& operator=(const Abonent& p);

Abonent<char\*>& operator=(Abonent&& p);

const char\* getName() const;

const char\* getSurname() const;

int getBookingDay() const;

int getBookingMonth() const;

int getBookingYear() const;

void setName(char\* name);

void setSafeName(char\* name);

void setSurname(char\* surname);

void setSafeSurname(char\* surname);

void setBookingDate(int dd, int mm, int yy);

void setSafeBookingDate(int dd, int mm, int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Abonent<char\*>& p);

};

template <>

class Abonent<std::string> : public BookingDate {

private:

std::string name;

std::string surname;

public:

Abonent(int dd = 1, int mm = 1, int yy = 1980,

std::string name = "Vasia", std::string surname = "Pupkin");

Abonent(const Abonent<std::string>& p);

Abonent(Abonent<std::string>&& p);

~Abonent();

Abonent<std::string>& operator=(const Abonent<std::string>& p);

std::string getName() const;

std::string getSurname() const;

int getBookingDay() const;

int getBookingMonth() const;

int getBookingYear() const;

void setName(std::string name);

void setSafeName(std::string name);

void setSurname(std::string surname);

void setSafeSurname(std::string surname);

void setBookingDate(int dd, int mm, int yy);

void setSafeBookingDate(int dd, int mm, int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Abonent<std::string>& p);

};

Abonent.cpp

#include "Abonent.h"

/\*

\* char\* implementation

\*/

Abonent<char\*>::Abonent(int dd, int mm, int yy, char\* n, char\* s) :

BookingDate(dd, mm, yy) {

if (strlen(n) <= 20) {

name = new char[strlen(n) + 1];

strcpy(name, n);

}

else {

name = new char[6];

strcpy(name, "Vasia");

}

if (strlen(s) <= 20) {

surname = new char[strlen(s) + 1];

strcpy(surname, s);

}

else {

surname = new char[7];

strcpy(surname, "Pupkin");

}

}

Abonent<char\*>::Abonent(const Abonent<char\*>& p) :BookingDate(p) {

name = new char[strlen(p.name) + 1];

strcpy(name, p.name);

surname = new char[strlen(p.surname) + 1];

strcpy(surname, p.surname);

}

Abonent<char\*>::Abonent(Abonent<char\*>&& p): BookingDate(std::move(p)) {

name = p.name;

surname = p.surname;

p.name = NULL;

p.surname = NULL;

}

Abonent<char\*>::~Abonent() {

if (name) {

delete[] name;

}

if (surname) {

delete[] surname;

}

}

const Abonent<char\*>& Abonent<char\*>::operator=(const Abonent& p) {

if (this == &p) {

return \*this;

}

BookingDate::operator=(p);

delete[] name;

name = new char[strlen(p.name) + 1];

strcpy(name, p.name);

delete[] surname;

surname = new char[strlen(p.surname) + 1];

strcpy(surname, p.surname);

return \*this;

}

Abonent<char\*>& Abonent<char\*>::operator=( Abonent&& p) {

if (this == &p) {

return \*this;

}

BookingDate::operator=(std::move(p));

if (name) {

delete[] name;

}

name = p.name;

p.name = NULL;

if (surname) {

delete[] surname;

}

surname = p.name;

p.surname = NULL;

return \*this;

}

const char\* Abonent<char\*>::getName() const {

return name;

}

const char\* Abonent<char\*>::getSurname() const {

return surname;

}

int Abonent<char\*>::getBookingDay() const {

return BookingDate::getDay();

}

int Abonent<char\*>::getBookingMonth() const {

return BookingDate::getMonth();

}

int Abonent<char\*>::getBookingYear() const {

return BookingDate::getYear();

}

void Abonent<char\*>::setName(char\* n) {

delete[] name;

name = new char[strlen(n) + 1];

strcpy(name, n);

}

void Abonent<char\*>::setSafeName(char\* n) {

if (strlen(n) <= 20) {

delete[] name;

name = new char[strlen(n) + 1];

strcpy(name, n);

}

}

void Abonent<char\*>::setSurname(char\* n) {

delete[] surname;

surname = new char[strlen(n) + 1];

strcpy(name, n);

}

void Abonent<char\*>::setSafeSurname(char\* n) {

if (strlen(n) <= 20) {

delete[] surname;

surname = new char[strlen(n) + 1];

strcpy(name, n);

}

}

void Abonent<char\*>::setBookingDate(int dd, int mm, int yy) {

BookingDate::setDate(dd, mm, yy);

}

void Abonent<char\*>::setSafeBookingDate(int dd, int mm, int yy) {

BookingDate::safeSetDate(dd, mm, yy);

}

std::ostream& operator<<(std::ostream& os, const Abonent<char\*>& p) {

char temp = os.fill();

os << "Abonent data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(20) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(20) << "Firstname";

os.put(179);

os << std::setw(20) << "Surname";

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(20) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(20) << p.getName();

os.put(179);

os << std::setw(20) << p.getSurname();

os.put(179);

os << std::setw(3) << p.getBookingDay();

os.put(179);

os << std::setw(5) << p.getBookingMonth();

os.put(179);

os << std::setw(4) << p.getBookingYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(20) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

/\*

\* string implementation

\*/

Abonent<std::string>::Abonent(int dd, int mm, int yy,

std::string n, std::string s) : BookingDate(dd, mm, yy) {

if (n.size() <= 20) {

name = n;

}

else {

name = "Vasia";

}

if (s.size() <= 20) {

surname = s;

}

else {

surname = "Pupkin";

}

}

Abonent<std::string>::Abonent(const Abonent<std::string>& p)

: BookingDate(p) {

name = p.name;

surname = p.surname;

}

Abonent<std::string>::~Abonent() {

}

Abonent<std::string>& Abonent<std::string>::operator=(const Abonent<std::string>& p) {

if (this == &p) {

return \*this;

}

BookingDate::operator=(p);

name = p.name;

surname = p.surname;

return \*this;

}

std::string Abonent<std::string>::getName() const {

return name;

}

std::string Abonent<std::string>::getSurname() const {

return surname;

}

int Abonent<std::string>::getBookingDay() const {

return BookingDate::getDay();

}

int Abonent<std::string>::getBookingMonth() const {

return BookingDate::getMonth();

}

int Abonent<std::string>::getBookingYear() const {

return BookingDate::getYear();

}

void Abonent<std::string>::setName(std::string n) {

name = n;

}

void Abonent<std::string>::setSafeName(std::string n) {

if (n.size() <= 20) {

name = n;

}

}

void Abonent<std::string>::setSurname(std::string s) {

surname = s;

}

void Abonent<std::string>::setSafeSurname(std::string s) {

if (s.size() <= 20) {

surname = s;

}

}

void Abonent<std::string>::setBookingDate(int dd, int mm, int yy) {

BookingDate::setDate(dd, mm, yy);

}

void Abonent<std::string>::setSafeBookingDate(int dd, int mm, int yy) {

BookingDate::safeSetDate(dd, mm, yy);

}

std::ostream& operator<<(std::ostream& os, const Abonent<std::string>& p) {

char temp = os.fill();

os << "Abonent data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(20) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(20) << "Firstname";

os.put(179);

os << std::setw(20) << "Surname";

os.put(179);

os << "Day";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(20) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(20) << p.getName();

os.put(179);

os << std::setw(20) << p.getSurname();

os.put(179);

os << std::setw(3) << p.getBookingDay();

os.put(179);

os << std::setw(5) << p.getBookingMonth();

os.put(179);

os << std::setw(4) << p.getBookingYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(20) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Abonent<char\*>::printText(std::ofstream& os) {

os << std::setw(20) << getName();

os.put(' ');

os << std::setw(20) << getSurname();

os.put(' ');

os << std::setw(3) << getBookingDay();

os.put(' ');

os << std::setw(5) << getBookingMonth();

os.put(' ');

os << std::setw(4) << getBookingYear();

os.put(' ');

os.put('\n');

}

void Abonent<std::string>::printText(std::ofstream& os) {

os << std::setw(20) << getName();

os.put(' ');

os << std::setw(20) << getSurname();

os.put(' ');

os << std::setw(3) << getBookingDay();

os.put(' ');

os << std::setw(5) << getBookingMonth();

os.put(' ');

os << std::setw(4) << getBookingYear();

os.put(' ');

os.put('\n');

}

void Abonent<char\*>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

void Abonent<std::string>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

Studio.h

#pragma once

#include "Date\_1.h"

#include <cstring>

#include <string>

template <class StrType>

class Studio : public FoundationDate {};

template<>

class Studio<char\*> : public FoundationDate {

private:

char\* studioName;

public:

Studio(short mm = 1, int yy = 1980, char\* name = "Default");

Studio(const Studio<char\*>& p);

Studio(Studio<char\*>&& p);

~Studio();

Studio<char\*>& operator=(const Studio<char\*>& p);

Studio<char\*>& operator=( Studio<char\*>&& p);

const char\* getStudioName() const;

short getFoundationMonth() const;

int getFoundationYear() const;

void setStudioName(char\* name);

void safeSetStudioName(char\* name);

void setSafeFoundationMonth(short mm);

void setSafeFoundationYear(int yy);

void setFoundationMonth(short mm);

void setFoundationYear(int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Studio<char\*>& p);

};

template<>

class Studio<std::string> : public FoundationDate {

private:

std::string studioName;

public:

Studio(short mm = 1, int yy = 1980, std::string name = "Default");

Studio(const Studio<std::string>& p);

~Studio();

Studio<std::string> operator=(const Studio<std::string>& p);

std::string getStudioName() const;

short getFoundationMonth() const;

int getFoundationYear() const;

void setStudioName(std::string name);

void safeSetStudioName(std::string name);

void setSafeFoundationMonth(short mm);

void setSafeFoundationYear(int yy);

void setFoundationMonth(short mm);

void setFoundationYear(int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

friend std::ostream& operator<<(std::ostream& os, const Studio<std::string>& p);

};

Studio.cpp

#include "Studio.h"

/\*

\*

\* Start char\* template implementation

\*

\*/

Studio<char\*>::Studio(short mm, int yy, char\* name) : FoundationDate(mm, yy) {

if (strlen(name) <= 25) {

studioName = new char[strlen(name) + 1];

strcpy(studioName, name);

}

else {

studioName = new char[8];

strcpy(studioName, "Default");

}

}

Studio<char\*>::Studio(const Studio<char\*>& p):FoundationDate(p) {

studioName = new char[strlen(p.studioName) + 1];

strcpy(studioName, p.studioName);

}

Studio<char\*>::Studio(Studio<char\*>&& p) : FoundationDate(std::move(p)) {

studioName = p.studioName;

p.studioName = NULL;

}

Studio<char\*>::~Studio() {

if (studioName) {

delete[] studioName;

}

}

Studio<char\*>& Studio<char\*>::operator=(const Studio<char\*>& p) {

if (this == &p) {

return \*this;

}

FoundationDate::operator=(p);

delete[] studioName;

studioName = new char[strlen(p.studioName) + 1];

strcpy(studioName, p.studioName);

return \*this;

}

Studio<char\*>& Studio<char\*>::operator=(Studio<char\*>&& p) {

if (this == &p) {

return \*this;

}

FoundationDate::operator=(p);

if (studioName) {

delete[] studioName;

}

studioName = p.studioName;

p.studioName = NULL;

return \*this;

}

const char\* Studio<char\*>::getStudioName() const {

return studioName;

}

short Studio<char\*>::getFoundationMonth() const {

return FoundationDate::getMonth();

}

int Studio<char\*>::getFoundationYear() const {

return FoundationDate::getYear();

}

void Studio<char\*>::setStudioName(char\* name) {

delete studioName;

studioName = new char[strlen(name) + 1];

strcpy(studioName, name);

}

void Studio<char\*>::safeSetStudioName(char\* name) {

if (strlen(name) <= 25) {

studioName = new char[strlen(name) + 1];

strcpy(studioName, name);

}

}

void Studio<char\*>::setSafeFoundationMonth(short mm) {

FoundationDate::safeSetMonth(mm);

}

void Studio<char\*>::setSafeFoundationYear(int yy) {

FoundationDate::safeSetYear(yy);

}

void Studio<char\*>::setFoundationMonth(short mm) {

FoundationDate::setMonth(mm);

}

void Studio<char\*>::setFoundationYear(int yy) {

FoundationDate::setYear(yy);

}

std::ostream& operator<<(std::ostream& os, const Studio<char\*>& p) {

char temp = os.fill();

os << "Studio data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(25) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(25) << "StudioName";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(25) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(5) << p.getFoundationMonth();

os.put(179);

os << std::setw(4) << p.getFoundationYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(25) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Studio<char\*>::printText(std::ofstream& os) {

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(5) << getFoundationMonth();

os.put(' ');

os << std::setw(4) << getFoundationYear();

os.put(' ');

os.put('\n');

}

void Studio<char\*>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

/\*

\*

\* End char\* template implementation

\*

\*/

/\*

\*

\* Start string template implementation

\*

\*/

Studio<std::string>::Studio(short mm, int yy, std::string name):FoundationDate(mm, yy) {

if (name.size() <= 25) {

studioName = name;

}

else {

studioName = "Default";

}

}

Studio<std::string>::Studio(const Studio<std::string>& p):FoundationDate(p) {

studioName = p.studioName;

}

Studio<std::string>::~Studio() {

}

Studio<std::string> Studio<std::string>::operator=(const Studio<std::string>& p) {

if (this == &p) {

return \*this;

}

studioName = p.studioName;

setFoundationMonth(p.getFoundationMonth());

setFoundationYear(p.getFoundationYear());

return \*this;

}

std::string Studio<std::string>::getStudioName() const {

return studioName;

}

short Studio<std::string>::getFoundationMonth() const {

return FoundationDate::getMonth();

}

int Studio<std::string>::getFoundationYear() const {

return FoundationDate::getYear();

}

void Studio<std::string>::setStudioName(std::string name) {

studioName = name;

}

void Studio<std::string>::safeSetStudioName(std::string name) {

if (name.size() <= 25) {

studioName = name;

}

}

void Studio<std::string>::setSafeFoundationMonth(short mm) {

FoundationDate::safeSetMonth(mm);

}

void Studio<std::string>::setSafeFoundationYear(int yy) {

FoundationDate::safeSetYear(yy);

}

void Studio<std::string>::setFoundationMonth(short mm) {

FoundationDate::setMonth(mm);

}

void Studio<std::string>::setFoundationYear(int yy) {

FoundationDate::setYear(yy);

}

std::ostream& operator<<(std::ostream& os, const Studio<std::string>& p) {

char temp = os.fill();

os << "Studio data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(25) << "";

os.put(194);

os << std::setw(5) << "";

os.put(194);

os << std::setw(4) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(25) << "StudioName";

os.put(179);

os << "Month";

os.put(179);

os << "Year";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(25) << "";

os.put(197);

os << std::setw(5) << "";

os.put(197);

os << std::setw(4) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(5) << p.getMonth();

os.put(179);

os << std::setw(4) << p.getYear();

os.put(179);

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(25) << "";

os.put(193);

os << std::setw(5) << "";

os.put(193);

os << std::setw(4) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Studio<std::string>::printText(std::ofstream& os) {

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(5) << getFoundationMonth();

os.put(' ');

os << std::setw(4) << getFoundationYear();

os.put(' ');

os.put('\n');

}

void Studio<std::string>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

/\*

\*

\* End string template implementation

\*

\*/

Disc.h

#pragma once

#include "Date\_2.h"

#include "Studio.h"

template <class StrType>

class Disc : public ProductionDate, public Studio<StrType> {

};

template <>

class Disc<char\*> : public ProductionDate, public Studio<char\*> {

private:

char\* filmName;

int duration;

public:

Disc(short mm = 1, int yy = 1980, char\* studioName = "Default",

char\* filmName = "Film", int duration = 0,

int prodDD = 1, int prodMM = 1, int prodYY = 1);

Disc(const Disc<char\*>& p);

Disc(Disc<char\*>&& p);

~Disc();

const Disc<char\*>& operator=(const Disc<char\*>& p);

Disc<char\*>& operator=(Disc<char\*>&& p);

const char\* getFilmName() const;

int getFilmDuration() const;

int getProductionDay() const;

int getProductionMonth() const;

int getProductionYear() const;

void setFilmName(char\* name);

void setSafeFilmName(char\* name);

void setFilmDuration(int val);

void setSafeFilmDuration(int val);

void setSafeProductionDate(int dd, int mm, int yy);

void setProductionDate(int dd, int mm, int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Disc<char\*>& p);

};

template <>

class Disc<std::string> : public ProductionDate, public Studio<std::string> {

private:

std::string filmName;

int duration;

public:

Disc(short mm = 1, int yy = 1980, std::string studioName = "Default",

std::string filmName = "Film", int duration = 0, int prodDD = 1,

int prodMM = 1, int prodYY = 1980);

Disc(const Disc<std::string>& p);

~Disc();

std::string getFilmName() const;

int getFilmDuration() const;

int getProductionDay() const;

int getProductionMonth() const;

int getProductionYear() const;

void setFilmName(std::string name);

void setSafeFilmName(std::string name);

void setFilmDuration(int val);

void setSafeFilmDuration(int val);

void setSafeProductionDate(int dd, int mm, int yy);

void setProductionDate(int dd, int mm, int yy);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

void printConsole() {

std::cout << \*this << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const Disc<std::string>& p);

};

Disc.cpp

#include "Disc.h"

/\*

\*

\* char\* implementation

\*

\*/

Disc<char\*>::Disc(short mm, int yy, char\* studioName,

char\* fName, int dur, int prodDD, int prodMM,

int prodYY):

Studio(mm, yy, studioName), ProductionDate(prodDD, prodMM, prodYY){

duration = dur;

if (strlen(fName) <= 40) {

filmName = new char[strlen(fName) + 1];

strcpy(filmName, fName);

}

else {

filmName = new char[5];

strcpy(filmName, "Film");

}

}

Disc<char\*>::Disc(const Disc<char\*>& p) : Studio(p), ProductionDate(p) {

duration = p.duration;

filmName = new char[strlen(p.filmName) + 1];

strcpy(filmName, p.filmName);

}

Disc<char\*>::Disc(Disc<char\*>&& p) : ProductionDate(std::move(p)), Studio(std::move(p)) {

filmName = p.filmName;

p.filmName = NULL;

duration = p.duration;

}

Disc<char\*>::~Disc() {

if (filmName) {

delete[] filmName;

}

}

const Disc<char\*>& Disc<char\*>::operator=(const Disc<char\*>& p) {

if (this == &p) {

return \*this;

}

Studio<char\*>::operator=(p);

ProductionDate::operator=(p);

duration = p.duration;

delete[] filmName;

filmName = new char[strlen(p.filmName) + 1];

strcpy(filmName, p.filmName);

return \*this;

}

Disc<char\*>& Disc<char\*>::operator=(Disc<char\*>&& p) {

if (this == &p) {

return \*this;

}

Studio<char\*>::operator=(p);

ProductionDate::operator=(p);

duration = p.duration;

if (filmName) {

delete[] filmName;

}

filmName = p.filmName;

p.filmName = NULL;

return \*this;

}

const char\* Disc<char\*>::getFilmName() const {

return filmName;

}

int Disc<char\*>::getFilmDuration() const {

return duration;

}

int Disc<char\*>::getProductionDay() const {

return ProductionDate::getDay();

}

int Disc<char\*>::getProductionMonth() const {

return ProductionDate::getMonth();

}

int Disc<char\*>::getProductionYear() const {

return ProductionDate::getYear();

}

void Disc<char\*>::setFilmName(char\* name) {

delete[] filmName;

filmName = new char[strlen(name) + 1];

strcpy(filmName, name);

}

void Disc<char\*>::setSafeFilmName(char\* name) {

if (strlen(name) <= 40) {

delete[] filmName;

filmName = new char[strlen(name) + 1];

strcpy(filmName, name);

}

}

void Disc<char\*>::setFilmDuration(int val) {

duration = val;

}

void Disc<char\*>::setSafeFilmDuration(int val) {

duration = (val > 0 && val < 1000) ? val : 0;

}

void Disc<char\*>::setProductionDate(int dd, int mm, int yy) {

ProductionDate::setDate(dd, mm, yy);

}

void Disc<char\*>::setSafeProductionDate(int dd, int mm, int yy) {

ProductionDate::safeSetDate(dd, mm, yy);

}

std::ostream& operator<<(std::ostream& os, const Disc<char\*>& p) {

char temp = os.fill();

os << "Disc data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(40) << "";

os.put(194);

os << std::setw(25) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(7) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << "Film";

os.put(179);

os << std::setw(25) << "Studio";

os.put(179);

os << "Dur";

os.put(179);

os << std::setw(10) <<"PDate";

os.put(179);

os << std::setw(7) << "FDate";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(40) << "";

os.put(197);

os << std::setw(25) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(7) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << p.getFilmName();

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(3) << p.getFilmDuration();

os.put(179);

os.fill('0');

os << std::setw(2) << p.getProductionDay() << '/'

<< std::setw(2) << p.getProductionMonth() << '/'

<< std::setw(4) << p.getProductionYear();

os.put(179);

os << std::setw(2) << p.getFoundationMonth() << '/'

<< std::setw(4) << p.getFoundationYear();

os.put(179);

os.put('\n');

os.fill(' ');

os.fill(196);

os.put(192);

os << std::setw(40) << "";

os.put(193);

os << std::setw(25) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(7) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Disc<char\*>::printText(std::ofstream& os) {

os.put(' ');

os << std::setw(40) << getFilmName();

os.put(' ');

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(3) << getFilmDuration();

os.put(' ');

os.fill('0');

os << std::setw(2) << getProductionDay() << '/'

<< std::setw(2) << getProductionMonth() << '/'

<< std::setw(4) << getProductionYear();

os.put(' ');

os << std::setw(2) << getFoundationMonth() << '/'

<< std::setw(4) << getFoundationYear();

os.put(' ');

os.fill(' ');

os.put('\n');

}

void Disc<char\*>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

/\*

\*

\* string implementation

\*

\*/

Disc<std::string>::Disc(short mm, int yy, std::string studioName,

std::string fName, int dur, int prodDD, int prodMM,

int prodYY) :Studio(mm, yy, studioName), ProductionDate(prodDD, prodMM, prodYY) {

duration = dur;

if (fName.size() <= 40) {

filmName = fName;

}

else {

filmName = "Film";

}

}

Disc<std::string>::Disc(const Disc<std::string>& p):Studio(p), ProductionDate(p) {

filmName = p.filmName;

duration = p.duration;

}

Disc<std::string>::~Disc() {

}

std::string Disc<std::string>::getFilmName() const {

return filmName;

}

int Disc<std::string>::getFilmDuration() const {

return duration;

}

int Disc<std::string>::getProductionDay() const {

return ProductionDate::getDay();

}

int Disc<std::string>::getProductionMonth() const {

return ProductionDate::getMonth();

}

int Disc<std::string>::getProductionYear() const {

return ProductionDate::getYear();

}

void Disc<std::string>::setFilmName(std::string name) {

filmName = name;

}

void Disc<std::string>::setSafeFilmName(std::string name) {

if (name.size() <= 45) {

filmName = name;

}

}

void Disc<std::string>::setFilmDuration(int val) {

duration = val;

}

void Disc<std::string>::setSafeFilmDuration(int val) {

duration = (val > 0) ? val : 0;

}

void Disc<std::string>::setProductionDate(int dd, int mm, int yy) {

ProductionDate::setDate(dd, mm, yy);

}

void Disc<std::string>::setSafeProductionDate(int dd, int mm, int yy) {

ProductionDate::safeSetDate(dd, mm, yy);

}

std::ostream& operator<<(std::ostream& os, const Disc<std::string>& p) {

char temp = os.fill();

os << "Disc data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(40) << "";

os.put(194);

os << std::setw(25) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(7) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << "Film";

os.put(179);

os << std::setw(25) << "Studio";

os.put(179);

os << "Dur";

os.put(179);

os << std::setw(10) << "PDate";

os.put(179);

os << std::setw(7) << "FDate";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(40) << "";

os.put(197);

os << std::setw(25) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(7) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << p.getFilmName();

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(3) << p.getFilmDuration();

os.put(179);

os.fill('0');

os << std::setw(2) << p.getProductionDay() << '/'

<< std::setw(2) << p.getProductionMonth() << '/'

<< std::setw(4) << p.getProductionYear();

os.put(179);

os << std::setw(2) << p.getFoundationMonth() << '/'

<< std::setw(4) << p.getFoundationYear();

os.put(179);

os.fill(' ');

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(40) << "";

os.put(193);

os << std::setw(25) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(7) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Disc<std::string>::printText(std::ofstream& os) {

os.put(' ');

os << std::setw(40) << getFilmName();

os.put(' ');

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(3) << getFilmDuration();

os.put(' ');

os.fill('0');

os << std::setw(2) << getProductionDay() << '/'

<< std::setw(2) << getProductionMonth() << '/'

<< std::setw(4) << getProductionYear();

os.put(' ');

os << std::setw(2) << getFoundationMonth() << '/'

<< std::setw(4) << getFoundationYear();

os.put(' ');

os.put('\n');

}

void Disc<std::string>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

#pragma once

enum Operation\_Type {RETURN, EXTRACT};

class OperationType {

private:

Operation\_Type type;

public:

OperationType(Operation\_Type t = RETURN);

OperationType(const OperationType& p);

Operation\_Type getType() const;

void setType(Operation\_Type t);

};

OperationType.cpp

#include "OperationType.h"

OperationType::OperationType(Operation\_Type t) : type(t) {}

OperationType::OperationType(const OperationType& p) { type = p.type; }

Operation\_Type OperationType::getType() const {

return type;

}

void OperationType::setType(Operation\_Type t) {

type = t;

}

Operation.h

#pragma once

#include "Disc.h"

#include "Abonent.h"

#include "OperationType.h"

#include <cstdlib>

#include <ctime>

char\* genString(const int len);

unsigned int getUnsigned(unsigned int max, unsigned int min = 0);

template <class StrType>

class Operation;

template <class StrType>

Operation<StrType> createRandomOperation(StrType nothing) {

StrType studioName = genString(25);

StrType filmName = genString(40);

StrType abName = genString(20);

StrType abSurname = genString(20);

Operation<StrType> temp(getUnsigned(12, 1), getUnsigned(9999, 1980), studioName, filmName,

getUnsigned(300, 0), getUnsigned(31, 1), getUnsigned(12, 1), getUnsigned(9999, 1980),

getUnsigned(31, 1), getUnsigned(12, 1), getUnsigned(9999, 1980), abName,

abSurname, RETURN, getUnsigned(31, 1), getUnsigned(12, 1), getUnsigned(9999, 1980));

return temp;

}

template <class StrType>

bool durationCmp(const Operation<StrType>& a, const Operation<StrType>& b);

template <class StrType>

class Operation : public Disc<StrType>, public Abonent<StrType>,

public OperationType, public OperationDate {

};

template<>

class Operation<char\*> : public Disc<char\*>, public Abonent<char\*>,

public OperationType, public OperationDate {

public:

Operation(short mm = 1, int yy = 1, char\* studio = "Studio",

char\* film = "Film", int dur = 0, int prodDD = 1, int prodMM = 1,

int prodYY = 1980, int abDD = 1, int abMM = 1, int abYY = 1980,

char\* abName = "Vasia", char\* abSurname = "Pupkin",

Operation\_Type type = RETURN, int opDD = 1,int opMM = 1,

int opYY = 1980 );

Operation(const Operation<char\*>& p);

Operation(Operation<char\*>&& p);

~Operation();

const Operation<char\*>& operator=(const Operation<char\*>& p);

bool operator>(const Operation<char\*>& p) const;

bool operator<(const Operation<char\*>& p) const;

int getOperationDay() const;

int getOperationMonth() const;

int getOperationYear() const;

void setOperationDate(int dd, int mm, int yy);

void setSafeOperationDate(int dd, int mm, int yy);

friend Operation<char\*> createRandomOperation<char\*>(char\* s);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

friend std::ostream& operator<<(std::ostream& os, const Operation<char\*>& p);

};

template<>

class Operation<std::string> : public Disc<std::string>, public Abonent<std::string>,

public OperationType, public OperationDate {

public:

Operation(short mm = 1, int yy = 1, std::string studio = "Studio",

std::string film = "Film", int dur = 0, int prodDD = 1, int prodMM = 1,

int prodYY = 1980, int abDD = 1, int abMM = 1, int abYY = 1980,

std::string abName = "Vasia", std::string abSurname = "Pupkin",

Operation\_Type type = RETURN, int opDD = 1, int opMM = 1,

int opYY = 1980);

Operation(const Operation<std::string>& p);

const Operation<std::string>& operator=(const Operation<std::string>& p);

bool operator>(const Operation<std::string>& p) const;

bool operator<(const Operation<std::string>& p) const;

int getOperationDay() const;

int getOperationMonth() const;

int getOperationYear() const;

void setOperationDate(int dd, int mm, int yy);

void setSafeOperationDate(int dd, int mm, int yy);

friend Operation<std::string> createRandomOperation<std::string>(std::string s);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

friend std::ostream& operator<<(std::ostream& os, const Operation<std::string>& p);

};

Operation.cpp

#include "Operation.h"

Operation<char\*>::Operation(short mm, int yy, char\* studio,

char\* film, int dur, int prodDD, int prodMM,

int prodYY, int abDD, int abMM, int abYY,

char\* abName, char\* abSurname,

Operation\_Type type, int opDD, int opMM,

int opYY) : Disc(mm, yy, studio, film, dur,prodDD, prodMM, prodYY),

Abonent(abDD, abMM, abYY, abName, abSurname), OperationType(type),

OperationDate(opDD, opMM, opYY){}

Operation<char\*>::Operation(const Operation<char\*>& p) :

Disc(p), Abonent(p), OperationType(p), OperationDate(p){}

Operation<char\*>::Operation(Operation<char\*>&& p) : Disc(std::move(p)), Abonent(std::move(p)), OperationDate(std::move(p)),

OperationType(std::move(p)) {

}

Operation<char\*>::~Operation(){}

const Operation<char\*>& Operation<char\*>::operator=(const Operation<char\*>& p) {

if (this == &p) {

return \*this;

}

Disc<char\*>::operator=(p);

Abonent<char\*>::operator=(p);

OperationType::operator=(p);

OperationDate::operator=(p);

return \*this;

}

bool Operation<char\*>::operator>(const Operation<char\*>& p) const {

return strcmp(getFilmName(), p.getFilmName()) > 0;

}

bool Operation<char\*>::operator<(const Operation<char\*>& p) const {

return strcmp(getFilmName(), p.getFilmName()) < 0;

}

int Operation<char\*>::getOperationDay() const {

return OperationDate::getDay();

}

int Operation<char\*>::getOperationMonth() const {

return OperationDate::getMonth();

}

int Operation<char\*>::getOperationYear() const {

return OperationDate::getYear();

}

void Operation<char\*>::setOperationDate(int dd, int mm, int yy) {

OperationDate::setDate(dd, mm, yy);

}

void Operation<char\*>::setSafeOperationDate(int dd, int mm, int yy) {

OperationDate::safeSetDate(dd, mm, yy);

}

Operation<std::string>::Operation(short mm, int yy, std::string studio,

std::string film, int dur, int prodDD, int prodMM,

int prodYY, int abDD, int abMM, int abYY,

std::string abName, std::string abSurname,

Operation\_Type type, int opDD, int opMM,

int opYY) : Disc(mm, yy, studio, film, dur, prodDD, prodMM, prodYY),

Abonent(abDD, abMM, abYY, abName, abSurname), OperationType(type),

OperationDate(opDD, opMM, opYY) {}

Operation<std::string>::Operation(const Operation<std::string>& p) :

Disc(p), Abonent(p), OperationType(p), OperationDate(p) {}

const Operation<std::string>& Operation<std::string>::operator=(const Operation<std::string>& p) {

if (this == &p) {

return \*this;

}

Disc<std::string>::operator=(p);

Abonent<std::string>::operator=(p);

OperationType::operator=(p);

OperationDate::operator=(p);

return \*this;

}

bool Operation<std::string>::operator>(const Operation<std::string>& p) const {

return getFilmName() > p.getFilmName();

}

bool Operation<std::string>::operator<(const Operation<std::string>& p) const {

return getFilmName() < p.getFilmName();

}

int Operation<std::string>::getOperationDay() const {

return OperationDate::getDay();

}

int Operation<std::string>::getOperationMonth() const {

return OperationDate::getMonth();

}

int Operation<std::string>::getOperationYear() const {

return OperationDate::getYear();

}

void Operation<std::string>::setOperationDate(int dd, int mm, int yy) {

OperationDate::setDate(dd, mm, yy);

}

void Operation<std::string>::setSafeOperationDate(int dd, int mm, int yy) {

OperationDate::safeSetDate(dd, mm, yy);

}

std::ostream& operator<<(std::ostream& os, const Operation<char\*>& p) {

char temp = os.fill();

os << "Operation data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(40) << "";

os.put(194);

os << std::setw(25) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(7) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << "Film";

os.put(179);

os << std::setw(25) << "Studio";

os.put(179);

os << "Dur";

os.put(179);

os << std::setw(20) << "Firstname";

os.put(179);

os << std::setw(20) << "Surname";

os.put(179);

os << std::setw(10) << "PDate";

os.put(179);

os << std::setw(10) << "BDate";

os.put(179);

os << std::setw(10) << "ODate";

os.put(179);

os << std::setw(7) << "FDate";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(40) << "";

os.put(197);

os << std::setw(25) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(7) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << p.getFilmName();

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(3) << p.getFilmDuration();

os.put(179);

os << std::setw(20) << p.getName();

os.put(179);

os << std::setw(20) << p.getSurname();

os.put(179);

os.fill('0');

os << std::setw(2) << p.getProductionDay() << '/'

<< std::setw(2) << p.getProductionMonth() << '/'

<< std::setw(4) << p.getProductionYear();

os.put(179);

os << std::setw(2) << p.getBookingDay() << '/'

<< std::setw(2) << p.getBookingMonth() << '/'

<< std::setw(4) << p.getBookingYear();

os.put(179);

os << std::setw(2) << p.getOperationDay() << '/'

<< std::setw(2) << p.getOperationMonth() << '/'

<< std::setw(4) << p.getOperationYear();

os.put(179);

os << std::setw(2) << p.getFoundationMonth() << '/'

<< std::setw(4) << p.getFoundationYear();

os.put(179);

os.fill(' ');

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(40) << "";

os.put(193);

os << std::setw(25) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(7) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

std::ostream& operator<<(std::ostream& os, const Operation<std::string>& p) {

char temp = os.fill();

os << "Operation data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(40) << "";

os.put(194);

os << std::setw(25) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(7) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << "Film";

os.put(179);

os << std::setw(25) << "Studio";

os.put(179);

os << "Dur";

os.put(179);

os << std::setw(20) << "Firstname";

os.put(179);

os << std::setw(20) << "Surname";

os.put(179);

os << std::setw(10) << "PDate";

os.put(179);

os << std::setw(10) << "BDate";

os.put(179);

os << std::setw(10) << "ODate";

os.put(179);

os << std::setw(7) << "FDate";

os.put(179);

os.put('\n');

os.fill(196);

os.put(195);

os << std::setw(40) << "";

os.put(197);

os << std::setw(25) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(7) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << p.getFilmName();

os.put(179);

os << std::setw(25) << p.getStudioName();

os.put(179);

os << std::setw(3) << p.getFilmDuration();

os.put(179);

os << std::setw(20) << p.getName();

os.put(179);

os << std::setw(20) << p.getSurname();

os.put(179);

os.fill('0');

os << std::setw(2) << p.getProductionDay() << '/'

<< std::setw(2) << p.getProductionMonth() << '/'

<< std::setw(4) << p.getProductionYear();

os.put(179);

os << std::setw(2) << p.getBookingDay() << '/'

<< std::setw(2) << p.getBookingMonth() << '/'

<< std::setw(4) << p.getBookingYear();

os.put(179);

os << std::setw(2) << p.getOperationDay() << '/'

<< std::setw(2) << p.getOperationMonth() << '/'

<< std::setw(4) << p.getOperationYear();

os.put(179);

os << std::setw(2) << p.getFoundationMonth() << '/'

<< std::setw(4) << p.getFoundationYear();

os.put(179);

os.fill(' ');

os.put('\n');

os.fill(196);

os.put(192);

os << std::setw(40) << "";

os.put(193);

os << std::setw(25) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(7) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

void Operation<char\*>::printText(std::ofstream& os) {

os.put(' ');

os << std::setw(40) << getFilmName();

os.put(' ');

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(3) << getFilmDuration();

os.put(' ');

os << std::setw(20) << getName();

os.put(' ');

os << std::setw(20) << getSurname();

os.put(' ');

os.fill('0');

os << std::setw(2) << getProductionDay() << '/'

<< std::setw(2) << getProductionMonth() << '/'

<< std::setw(4) << getProductionYear();

os.put(' ');

os << std::setw(2) << getBookingDay() << '/'

<< std::setw(2) << getBookingMonth() << '/'

<< std::setw(4) << getBookingYear();

os.put(' ');

os << std::setw(2) << getOperationDay() << '/'

<< std::setw(2) << getOperationMonth() << '/'

<< std::setw(4) << getOperationYear();

os.put(' ');

os << std::setw(2) << getFoundationMonth() << '/'

<< std::setw(4) << getFoundationYear();

os.put(' ');

os.fill(' ');

os.put('\n');

}

void Operation<std::string>::printText(std::ofstream& os) {

os.put(' ');

os << std::setw(40) << getFilmName();

os.put(' ');

os << std::setw(25) << getStudioName();

os.put(' ');

os << std::setw(3) << getFilmDuration();

os.put(' ');

os << std::setw(20) << getName();

os.put(' ');

os << std::setw(20) << getSurname();

os.put(' ');

os.fill('0');

os << std::setw(2) << getProductionDay() << '/'

<< std::setw(2) << getProductionMonth() << '/'

<< std::setw(4) << getProductionYear();

os.put(' ');

os << std::setw(2) << getBookingDay() << '/'

<< std::setw(2) << getBookingMonth() << '/'

<< std::setw(4) << getBookingYear();

os.put(' ');

os << std::setw(2) << getOperationDay() << '/'

<< std::setw(2) << getOperationMonth() << '/'

<< std::setw(4) << getOperationYear();

os.put(' ');

os << std::setw(2) << getFoundationMonth() << '/'

<< std::setw(4) << getFoundationYear();

os.put(' ');

os.fill(' ');

os.put('\n');

}

void Operation<char\*>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

void Operation<std::string>::printBinary(std::ofstream& os) {

os.write(reinterpret\_cast<const char\*>(this), sizeof(\*this));

}

OperationList.h

#pragma once

#include "Operation.h"

#include <cassert>

template <class T, class StrType>

class OperationList;

template <class T, class StrType>

int getAverageDuration(const OperationList<T, StrType>& list);

template <class T, class StrType>

std::ostream& operator<< (std::ostream& os, const OperationList<T, StrType>& list);

template <class T, class StrType>

class OperationList : public OperationListDate {

private:

T size;

T quantity;

Operation<StrType>\* pointer;

public:

OperationList(T size = 10, int dd = 1, int mm = 1,int yy = 1980);

OperationList(const OperationList<T, StrType>& p);

~OperationList();

const OperationList<T, StrType>& operator=(const OperationList<T, StrType>& p);

const Operation<StrType> operator[](T index) const;

T getSize() const;

T getQuantity() const;

int getOperationListDay() const;

int getOperationListMonth() const;

int getOperationListYear() const;

void setOperationListDate(int dd, int mm, int yy);

void setSafeOperationListDate(int dd, int mm, int yy);

void insert(const Operation<StrType>& item);

void fill();

void sortByName();

void sortByDuration();

void sortByNameSw();

void sortByDurationSw();

friend int getAverageDuration<>(const OperationList<T, StrType>& list);

friend std::ostream& operator<<<>(std::ostream& os, const OperationList<T, StrType>& list);

};

template <class T, class StrType>

OperationList<T, StrType>::OperationList(T s, int dd, int mm, int yy) :

OperationListDate(dd, mm, yy) {

size = s;

quantity = 0;

pointer = new Operation<StrType>[s];

}

template <class T, class StrType>

OperationList<T, StrType>::OperationList(const OperationList& p) :

OperationListDate(p) {

size = p.size;

quantity = p.quantity;

pointer = new Operation<StrType>[size];

for (T i = 0; i < quantity; i++) {

pointer[i] = p[i];

}

}

template <class T, class StrType>

OperationList<T, StrType>::~OperationList() {

delete[] pointer;

}

template <class T, class StrType>

const OperationList<T, StrType>& OperationList<T, StrType>::operator=(const OperationList<T, StrType>& p) {

if (this == &p) {

return \*this;

}

delete[] pointer;

size = p.size;

quantity = p.quantity;

pointer = new Operation[size];

for (T i = 0; i < quantity; i++) {

pointer[i] = p[i];

}

return \*this;

}

template <class T, class StrType>

const Operation<StrType> OperationList<T, StrType>::operator[](T index) const {

assert(index >= 0 && index < quantity);

return pointer[index];

}

template <class T, class StrType>

T OperationList<T, StrType>::getSize() const {

return size;

}

template <class T, class StrType>

T OperationList<T, StrType>::getQuantity() const {

return quantity;

}

template <class T, class StrType>

int OperationList<T, StrType>::getOperationListDay() const {

return OperationListDate::getDay();

}

template <class T, class StrType>

int OperationList<T, StrType>::getOperationListMonth() const {

return OperationListDate::getMonth();

}

template <class T, class StrType>

int OperationList<T, StrType>::getOperationListYear() const {

return OperationListDate::getYear();

}

template <class T, class StrType>

void OperationList<T, StrType>::setOperationListDate(int dd, int mm, int yy) {

OperationListDate::setDate(dd, mm, yy);

}

template <class T, class StrType>

void OperationList<T, StrType>::setSafeOperationListDate(int dd, int mm, int yy) {

OperationListDate::setSafeDate(dd, mm, yy);

}

template <class T, class StrType>

void OperationList<T, StrType>::insert(const Operation<StrType>& item) {

assert(quantity < size);

pointer[quantity++] = item;

}

template <class T, class StrType>

void OperationList<T, StrType>::fill() {

StrType temp = "1";

for (T i = quantity; i < size; i++) {

insert(createRandomOperation(temp));

}

}

template <class T, class StrType>

void OperationList<T, StrType>::sortByName() {

/\* Selection Sort \*/

for (T i = 0; i < quantity - 1; i++) {

for (T j = i + 1; j < quantity; j++) {

if (pointer[i] > pointer[j]) {

Operation<StrType> temp = pointer[i];

pointer[i] = pointer[j];

pointer[j] = temp;

}

}

}

}

template <class T, class StrType>

void OperationList<T, StrType>::sortByDuration() {

/\* Selection Sort \*/

for (T i = 0; i < quantity - 1; i++) {

for (T j = i + 1; j < quantity; j++) {

if (pointer[i].getFilmDuration() > pointer[j].getFilmDuration()) {

Operation<StrType> temp = pointer[i];

pointer[i] = pointer[j];

pointer[j] = temp;

}

}

}

}

template <class T, class StrType>

void OperationList<T, StrType>::sortByNameSw() {

/\* Selection Sort \*/

for (T i = 0; i < quantity - 1; i++) {

for (T j = i + 1; j < quantity; j++) {

if (pointer[i] > pointer[j]) {

std::swap(pointer[i], pointer[j]);

}

}

}

}

template <class T, class StrType>

void OperationList<T, StrType>::sortByDurationSw() {

/\* Selection Sort \*/

for (T i = 0; i < quantity - 1; i++) {

for (T j = i + 1; j < quantity; j++) {

if (pointer[i].getFilmDuration() > pointer[j].getFilmDuration()) {

std::swap(pointer[i], pointer[j]);

}

}

}

}

///////////////////////////////////////////////////////////////////////////

template <class T, class StrType>

int getAverageDuration(const OperationList<T, StrType>& list) {

float sum = 0;

for (T i = 0; i < list.quantity; i++) {

sum = sum + (list[i]).getFilmDuration();

}

return sum / list.quantity;

}

template <class T, class StrType>

std::ostream& operator<< (std::ostream& os, const OperationList<T, StrType>& list) {

char temp = os.fill();

os << "OperationList data:" << std::endl;

os.fill(196);

os.put(218);

os << std::setw(40) << "";

os.put(194);

os << std::setw(25) << "";

os.put(194);

os << std::setw(3) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(20) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(10) << "";

os.put(194);

os << std::setw(7) << "";

os.put(191);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << "Film";

os.put(179);

os << std::setw(25) << "Studio";

os.put(179);

os << "Dur";

os.put(179);

os << std::setw(20) << "Firstname";

os.put(179);

os << std::setw(20) << "Surname";

os.put(179);

os << std::setw(10) << "PDate";

os.put(179);

os << std::setw(10) << "BDate";

os.put(179);

os << std::setw(10) << "ODate";

os.put(179);

os << std::setw(7) << "FDate";

os.put(179);

os.put('\n');

for (T i = 0; i < list.quantity; i++) {

os.fill(196);

os.put(195);

os << std::setw(40) << "";

os.put(197);

os << std::setw(25) << "";

os.put(197);

os << std::setw(3) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(20) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(10) << "";

os.put(197);

os << std::setw(7) << "";

os.put(180);

os.put('\n');

os.fill(' ');

os.put(179);

os << std::setw(40) << list[i].getFilmName();

os.put(179);

os << std::setw(25) << list[i].getStudioName();

os.put(179);

os << std::setw(3) << list[i].getFilmDuration();

os.put(179);

os << std::setw(20) << list[i].getName();

os.put(179);

os << std::setw(20) << list[i].getSurname();

os.put(179);

os.fill('0');

os << std::setw(2) << list[i].getProductionDay() << '/'

<< std::setw(2) << list[i].getProductionMonth() << '/'

<< std::setw(4) << list[i].getProductionYear();

os.put(179);

os << std::setw(2) << list[i].getBookingDay() << '/'

<< std::setw(2) << list[i].getBookingMonth() << '/'

<< std::setw(4) << list[i].getBookingYear();

os.put(179);

os << std::setw(2) << list[i].getOperationDay() << '/'

<< std::setw(2) << list[i].getOperationMonth() << '/'

<< std::setw(4) << list[i].getOperationYear();

os.put(179);

os << std::setw(2) << list[i].getFoundationMonth() << '/'

<< std::setw(4) << list[i].getFoundationYear();

os.put(179);

os.fill(' ');

os.put('\n');

}

os.fill(196);

os.put(192);

os << std::setw(40) << "";

os.put(193);

os << std::setw(25) << "";

os.put(193);

os << std::setw(3) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(20) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(10) << "";

os.put(193);

os << std::setw(7) << "";

os.put(217);

os.put('\n');

os.fill(temp);

return os;

}

CustomArray.h

#pragma once

#include <typeinfo>

#include <iostream>

#include <cassert>

template <class Type>

class CustomArray {

private:

Type\* pointer;

int size;

int quantity;

public:

CustomArray(int size = 10);

CustomArray(const CustomArray<Type>& p);

~CustomArray();

const CustomArray<Type>& operator=(const CustomArray<Type>& p);

Type& operator[] (int index) const;

int getSize() const;

int getQuantity() const;

void insert(const Type& item);

void printText(std::ofstream& descr);

void printBinary(std::ofstream& descr);

};

template <class Type>

CustomArray<Type>::CustomArray(int s) {

if (s < 1) { s = 10; }

pointer = new Type[s];

quantity = 0;

size = s;

}

template <class Type>

CustomArray<Type>::CustomArray(const CustomArray<Type>& p) {

size = p.size;

quantity = p.quantity;

pointer = new Type[size];

for (int i = 0; i < quantity; i++) {

pointer[i] = p.pointer[i];

}

}

template <class Type>

CustomArray<Type>::~CustomArray() {

delete[] pointer;

}

template <class Type>

const CustomArray<Type>& CustomArray<Type>::operator=(const CustomArray<Type>& p) {

if (this == &p) {

return \*this;

}

delete pointer;

size = p.size;

quantity = p.quantity;

pointer = new Type[size];

for (int i = 0; i < quantity; i++) {

pointer[i] = p.pointer[i];

}

return \*this;

}

template <class Type>

Type& CustomArray<Type>::operator[](int index) const {

assert(index >= 0 && index < quantity);

return pointer[index];

}

template <class Type>

int CustomArray<Type>::getSize() const {

return size;

}

template <class Type>

int CustomArray<Type>::getQuantity() const {

return quantity;

}

template <class Type>

void CustomArray<Type>::insert(const Type& p) {

assert(quantity < size);

pointer[quantity++] = p;

}

template <class Type>

void CustomArray<Type>::printText(std::ofstream& descr) {

for (int i = 0; i < quantity; i++) {

pointer[i].printText(descr);

}

}

template <class Type>

void CustomArray<Type>::printBinary(std::ofstream& descr) {

for (int i = 0; i < quantity; i++) {

pointer[i].printBinary(descr);

}

}

1. Шаблони класів – схожі на шаблони функцій, відрізняючись від них способом реалізації, а саме для створення шаблонів функції вона викликається з аргументами потрібного типу, а класи реалізуються за допомогою визначеного об’єкту, що використовує шаблонний аргумент ( Приклад наявний в коді програми для лабораторної роботи).

2. Множинне успадковування – за принципом успадковування є можливим множинне успадковування, а саме, коли деякий клас може успадковувати компоненти декількох ніяк не пов’язаних між собою класів. Коли успадковано декілька класів конструктори викликаються зліва направо (так як було написано в оголошенні класу.)

3. Низькорівневі процеси введення/виведення – це такі процеси вводу/ виводу, які використовують безпосередньо команді операційної системи, не підтримують форматування та буферизації.

4. Для потокового вводу/виводу перевантажена операція побітового зсуву

class Date

{

int mo, da, yr;

public:

Date(int m, int d, int y)

{

mo = m; da = d; yr = y;

}

friend ostream& operator<<(ostream& os, const Date& dt);

};

ostream& operator<<(ostream& os, const Date& dt)

{

os << dt.mo << '/' << dt.da << '/' << dt.yr;

return os;

}

5. Відкрити файл для зчитування за допомогою ifstream можна за допомогою прапорця ios::in

6. seekg(): Перший аргумент це зміщення відносно позиції заданої в другому аргументі, заданого за допомогою перерахування ios\_base::seekdir.