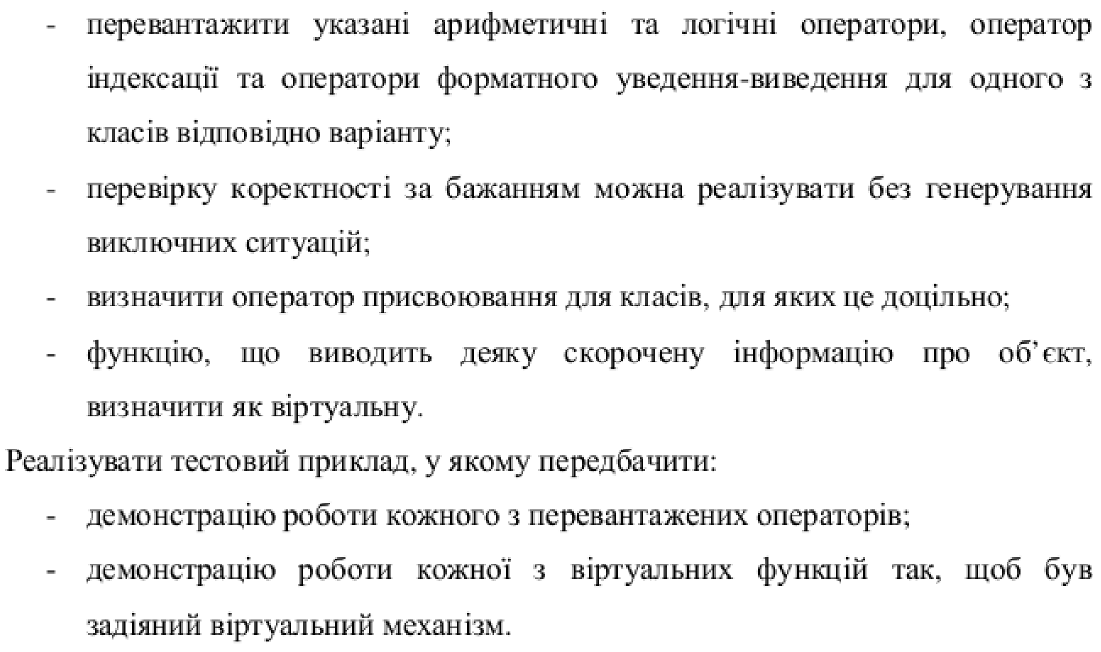
Умова



- перевантажити указані арифметичні та логічні оператори, оператор індексації та оператори форматного уведення-виведення для одного з класів відповідно варіанту;
- перевірку коректності за бажанням можна реалізувати без генерування виключних ситуацій;
- визначити оператор присвоювання для класів, для яких це доцільно;
- функцію, що виводить деяку скорочену інформацію про об’єкт, визначити як віртуальну.

Реалізувати тестовий приклад, у якому передбачити:

- демонстрацію роботи кожного з перевантажених операторів;
- демонстрацію роботи кожної з віртуальних функцій так, щоб був задіяний віртуальний механізм.

Визначено:
- арифметичний оператор “+” для класу Date;
- логічні оператори “<”та “>” за вартістю робіт – для класу “Дослідження”;
- оператор індексації для доступу до інформації про публікацію – для класу “Дослідження”;
- логічні оператори “==”та “!=” для перевірки збігу двох об’єктів класу “Студент”;
- оператори форматного уведення-виведення – для класів “Студент” та “Публікація”;
-опреатор присвоювання – для класів Студент, Замовник, Публікація, Дослідження.

Текст програми

lab1mod.hpp

```
#include <iostream> //just for defining a types std::ostream& and
std::istream&
class Date{
   private:
      int day, month, year;
   public:
      Date();
       Date(int, int, int);
      Date(const Date&);
```

```
        int getDay() const;
        int getMonth() const;
        int getYear() const;
        char *out() const;
        Date operator+(const Date&);
        Date& modDay(int);
        Date& modMonth(int);
        Date& modYear(int);
        ~Date();
        static bool verify(const int*);
};

class Student{
    private:
        char *name, *surname;
        int enroll_year;
    public:
        Student();
        Student(const char*, const char*, int);
        Student(const Student&);
        const char* getName() const;
        const char* getSurname() const;
        int getEnrollYear() const;
        char* out() const;
        Student& modName(const char *);
        Student& modSurname(const char *);
        Student& modEnrollYear(int);
        ~Student();
        bool operator==(const Student&);
        bool operator!=(const Student&);
        friend std::ostream& operator <<(std::ostream&, const Student&);
        friend std::istream& operator >>(std::istream&, Student&);
        Student& operator=(const Student& rhs);
};

//полиморфные классы допускают обработку объектов, тип которых
неизвестен во время компиляции
class Customer{
    protected:
        char *name, *theme;
        int price;
    public:
        Customer();
        Customer(const char*, const char*, int);
        Customer(const Customer&);
        const char* getName() const;
        const char* getTheme() const;
        int getPrice() const;
        virtual char *out() const; // if virtual is removed from the declaration, in all
cases the version of the base
                                //class would  have been called instead
        //Функции, описанные в базовом классе как виртуальные, могут быть
```

```
модифицированы в производных классах, причем связывание
        //произойдет не на этапе компиляции (то, что называется ранним
связыванием), а в момент обращения к
        //данному методу (позднее связывание).
        Customer& modName(const char*);
        Customer& modTheme(const char*);
        Customer& modPrice(int);
        virtual ~Customer();
        Customer& operator=(const Customer&);
};

enum SciAchivment {THESIS, ARTICLE, REPORT, INTARTICLE};

class Publication{
    private:
        Student* author;
        SciAchivment pub_type;
    public:
        Publication();
        Publication(const Student&, const SciAchivment);
        Publication(const Publication&);
        const Student& getAuthor() const;
        SciAchivment getPublicationType() const;
        char * out() const;
        Publication& modAuthor(const Student&);
        Publication& modPublicationType(const SciAchivment);
        ~Publication();
        friend std::ostream& operator<<(std::ostream&, const Publication&);
        friend std::istream& operator>>(std::istream&, Publication&);
        Publication& operator=(const Publication&);
};

class Research: public Customer{
    private:
        Date* sign_date;
        Publication** publications;
        int num_of_publications;
    public:
        Research();
        Research(const Customer&, const Date&);
        Research(const Research&);
        bool operator>(const Research&);
        bool operator<(const Research&);
        const Date& getSignDate() const;
        int getNumOfPublications() const;
        const Publication* const* getPublicationList() const;
        char * getInfo() const;
        char *out() const;
        Research& addPublication(const Publication&);
        Research& modCustomer(const Customer&);
        char const* operator[](int);
        ~Research();
```

```
	Research& operator=(const Research&);
};
```

lab1mod.cpp

```
#include "lab1mod.hpp"
#include <stdio.h>
#include <ctime>
#include <cmath>
#include <cstring>
#include <cstdlib>
#include <regex>
#include <iostream>
//For debugging
//#include <iostream>

//------------------------------------------------

//CONSTRUCTORS DATE

Date::Date(){
    //std::time_t time( std::time_t* arg ) - returns the current calendar time
encoded as a std::time_t object, and also stores it
    //in the object pointed to by arg, unless arg is a null pointer.
    std::time_t now = std::time(NULL);
    //struct tm* - pointer on time structure; structure containing a calendar date
and time broken down into its components.
    //struct tm * localtime (const time_t * timer) - uses the value pointed by
timer to fill a tm structure with the values that
    //represent the corresponding time, expressed for the local timezone.
    struct tm* tstruct = std::localtime(&now);
    //and now aggign values from received structure-calendar
    this->day = tstruct->tm_mday;
    this->month = tstruct->tm_mon + 1;
    this->year = tstruct->tm_year + 1900;

};

Date::Date(int in_day, int in_month, int in_year){
    this->day = in_day;
    this->month = in_month;
    this->year = in_year;

}

//constructor of copy
Date::Date(const Date& in_date){
    this->day = in_date.getDay(); //as fields day, month, year are private; we
can change 'em only with the class methods
    this->month = in_date.getMonth();
    this->year = in_date.getYear();
```

```
}

//CONSTRUCTORS DATE

//DESTRUCTOR DATE
Date::~Date(){

}


int Date::getDay() const{
    return this->day;
}

int Date::getMonth() const{
    return this->month;
}

int Date::getYear() const{
    return this->year;
}

//date output in string representation
char * Date::out() const{
    char * res = new char[10];
    sprintf(res, "%02i.%02i.%04i", this->day, this->month, this->year);
    return res;
}

//modifications of day, month, year
Date& Date::modDay(int inday){
    this->day = inday;
    return *this;
}

Date & Date::modMonth(int in_month){
    this->month = in_month;
    return *this;
}

Date& Date::modYear(int in_year){
    this->year = in_year;
    return *this;
}
//

//check whether the date is correct
bool Date::verify(const int * in_date){
    if (! (1582<= *(in_date + 2) )  )
        return false;
    if (! (1<= *(in_date + 1) && *(in_date + 1)<=12) )
```

```
        return false;
    if (! (1<= *(in_date) && *(in_date)<=31) )
        return false;
    if ( (*(in_date)==31) && (*(in_date + 1)==2 || *(in_date + 1)==4 || *(in_date
+ 1)==6 || *(in_date + 1)==9 || *(in_date + 1)==11) )
        return false;
    if ( (*(in_date)==30) && (*(in_date + 1)==2) )
        return false;
    if ( (*(in_date + 1)==2) && (*(in_date)==29) && (*(in_date + 2)%4!=0) )
        return false;
    if ( (*(in_date + 1)==2) && (*(in_date)==29) && (*(in_date + 2)%400==0) )
        return true;
    if ( (*(in_date + 1)==2) && (*(in_date)==29) && (*(in_date + 2)%100==0) )
        return false;
    if ( (*(in_date + 1)==2) && (*(in_date)==29) && (*(in_date + 2)%4==0)  )
        return true;
    return true;
}

Date Date::operator+(const Date& date_exapmle){
        Date temp_date;
        temp_date.day = this->day+date_exapmle.day;
        temp_date.month = this->month+date_exapmle.month;
        temp_date.year = this->year+date_exapmle.year;
        int* temp_array = new int[3];
        temp_array[0]=temp_date.day; temp_array[1]=temp_date.month;
temp_array[2]=temp_date.year;
        if (temp_date.verify(temp_array)){delete temp_array; return temp_date;}
        else {delete temp_array; return date_exapmle;}
}


//------------------------------------------------

//------------------------------------------------

//CONSTRUCTORS STUDENT
Student::Student(){
    this->name = new char[sizeof "Name"];
    this->surname = new char[sizeof "Surname"];
    std::strcpy(this->name, "Name");
    std::strcpy(this->surname, "Surname");
    //
    std:: time_t now = std::time(NULL);
    struct tm *tstruct = std::localtime(&now);
    //
    this->enroll_year = 1900 + tstruct->tm_year;
    return ;
}

Student::Student(const char * in_name, const char * in_surname, int in_year){
    this->name = new char[sizeof in_name];
    this->surname = new char[sizeof in_surname];
    std::strcpy(this->name, in_name);
    std::strcpy(this->surname, in_surname);
    this->enroll_year = in_year;
    return ;
}

//constructor of copy
Student::Student(const Student& in_student){
    this->name = new char[sizeof in_student.name];
    this->surname = new char[sizeof in_student.surname];
    std::strcpy(this->name, in_student.name);
    std::strcpy(this->surname, in_student.surname);
    this->enroll_year = in_student.enroll_year;
    return ;
}
//CONSTRUCTORS STUDENT

//DESTRUCTOR STUDENT
Student::~Student(){
    //free memory
    delete[] this->name;
    delete[] this->surname;
    return ;
}

const char * Student::getName() const{
    return (const char *)this->name;
}

const char * Student::getSurname() const{
    return (const char *)this->surname;
}

int Student::getEnrollYear() const{
    return this->enroll_year;
}

//output full info about student
char * Student::out() const{
    char * res = new char[(std::strlen(this->name) + 1) + (std::strlen(this-
>surname) + 1) + 6];
    sprintf(res, "%s %s\n%04i",
                    this->name, this->surname, this->enroll_year); //sends
formatted output to a string res
    return res;
}

//modife student's name
Student& Student::modName(const char * in_name){
    delete[] this->name;
    this->name = new char[sizeof in_name];
```

```
   std::strcpy(this->name, in_name);
   return *this;
}

Student& Student::modSurname(const char * in_surname){
   delete[] this->surname;
   this->surname = new char[sizeof in_surname];
   std::strcpy(this->surname, in_surname);
   return *this;
}

Student & Student::modEnrollYear(int inyear){
   this->enroll_year = inyear;
   return *this;
}

bool Student::operator==(const Student& rhs){
            return (strcmp(this->getName(), rhs.getName())==0 && strcmp(this-
>getSurname(), rhs.getSurname())==0
                                                && this-
>getEnrollYear()==rhs.getEnrollYear());
}

bool Student::operator!=(const Student& rhs){
      return !((*this)==rhs);
}


std::ostream& operator<<(std::ostream& os, const Student& rhs) {
       os << "==Name==" << rhs.getName() << "/==Surname== " <<
rhs.getSurname()
              << "/Enroll year: " << rhs.getEnrollYear()<< std::endl;
       return os;
}


std::istream& operator>>(std::istream& is, Student& rhs) {
       is.clear();
      is.ignore(80, '\n');
      char* name = new char[20];
      char* surname = new char[20];
      std::cout << "Input name:\n";
      is >> name;
      std::cout << "Input surname:\n";
      is >> surname;
       int EnrollYear;
      std::cout << "Input enroll year: ";
      is >> EnrollYear;
      rhs.modName(name);
      rhs.modSurname(surname);
      int * test = new int;
      *test = (int)EnrollYear;
```

```
        if (1+*test && *test>1582) rhs.modEnrollYear(EnrollYear);
        else rhs.modEnrollYear(0);
        is.clear();
         return is;
}

Student& Student::operator=(const Student& rhs){
        if (this == &rhs) return *this; // Gracefully handle self assignment
        delete[] name;
        delete[] surname;
        name = new char[sizeof(rhs.name)];
        surname = new char[sizeof(rhs.surname)];
        strcpy(name, rhs.name);
        strcpy(surname, rhs.surname);
        enroll_year=rhs.enroll_year;
        return *this;
}
//-------------------------------------------------

//-------------------------------------------------

//CONSTRUCTORS CUSTOMER

Customer::Customer(){
    this->name = new char[sizeof "Name"];
    this->theme = new char[sizeof "Theme"];
    std::strcpy(this->name, "Name");
    std::strcpy(this->theme, "Theme");
    this->price = 0;

}

Customer::Customer(const char * in_name, const char * in_theme, int in_price)
{
    this->name = new char[sizeof in_name];
    this->theme = new char[sizeof in_theme];
    std::strcpy(this->name, in_name);
    std::strcpy(this->theme, in_theme);
    this->price = in_price;

}

//constructor of copy
Customer::Customer(const Customer& in_research){
    this->name = new char[sizeof in_research.name];
    this->theme = new char[sizeof in_research.theme];
    std::strcpy(this->name, in_research.name);
    std::strcpy(this->theme, in_research.theme);
    this->price = in_research.price;

}
//CONSTRUCTORS CUSTOMER
```

```
//DESTRUCTOR CUSTOMER
Customer::~Customer(){
    delete[] this->name;
    delete[] this->theme;

}

const char * Customer::getName() const{
    return (const char *)this->name;
}

const char * Customer::getTheme() const{
    return (const char *)this->theme;
}

int Customer::getPrice() const{
    return this->price;
}

//output full info about customer
char * Customer::out() const{
    char * res = new char[(std::strlen(this->name) + 1) + (std::strlen(this-
>theme) + 1) + 6];
    sprintf(res, "%s %s\n%04i",
                    this->name, this->theme, this->price);
    return res;
}

Customer & Customer::modName(const char * in_name){
    delete[] this->name;
    this->name = new char[sizeof in_name];
    std::strcpy(this->name, in_name);
    return *this;
}

Customer & Customer::modTheme(const char * in_theme){
    delete[] this->theme;
    this->theme = new char[sizeof in_theme];
    std::strcpy(this->theme, in_theme);
    return *this;
}

Customer & Customer::modPrice(int in_price){
    this->price = in_price;
    return *this;
}

Customer& Customer::operator=(const Customer& rhs){
        if (this == &rhs) return *this; // Gracefully handle self assignment
        delete[] name;
        delete[] theme;
        name = new char[sizeof(rhs.name)];
        theme = new char[sizeof(rhs.theme)];
        strcpy(name, rhs.name);
        strcpy(theme, rhs.theme);
        price=rhs.price;
        return *this;
}
//------------------------------------------------

//------------------------------------------------

//CONSTRUCTORS PUBLICATION

Publication::Publication(){
    this->author = new Student();
    this->pub_type = THESIS;

};

Publication::Publication(const Student& in_author, const SciAchivment
in_pub_type){
    this->author = new Student(in_author);//initialize Student with some name
in_author
    this->pub_type = in_pub_type;

};

//constructor of copy
Publication::Publication(const Publication& in_publication){
    this->author = new Student(in_publication.getAuthor());
    this->pub_type = in_publication.getPublicationType();

};

//CONSTRUCTORS PUBLICATION

//get author
const Student& Publication::getAuthor() const{
    return *(this->author);
};

//get publication type
SciAchivment Publication::getPublicationType() const{
    return this->pub_type;
};

//output all info about Publication
char * Publication::out() const{
    char * a_name = this->author->out();
    char * p_name;
    switch(this->pub_type){
        case THESIS : p_name = new char[sizeof "thesis for report"];
```

```
                  std::strcpy(p_name, "thesis for report");
                  break;
       case ARTICLE : p_name = new char[sizeof "article in proffesional journal"];
                   std::strcpy(p_name, "article in proffesional journal");
                   break;
       case REPORT : p_name = new char[sizeof "report on conference"];
                  std::strcpy(p_name, "report on conference");
                  break;
       case INTARTICLE : p_name = new char[sizeof "article in an international
science journal"];
                      std::strcpy(p_name, "article in an international science
journal");
                      break;
    };
    char * res = new char[(std::strlen(a_name) + 1) + (std::strlen(p_name) + 1)
+ sizeof "\nPublication type : "];
    std::strcpy(res, a_name);
    std::strcat(res, "\tPublication type : ");
    std::strcat(res, p_name);
    delete[] p_name;
    delete[] a_name;
    return res;
};

//change author
Publication& Publication::modAuthor(const Student & in_author){
    delete this->author;
    this->author = new Student(in_author);
    return *this;
};

//change publication type
Publication& Publication::modPublicationType(const SciAchivment in_pub_type)
{
    this->pub_type = in_pub_type;
    return *this;
};

//DESTRUCTOR PUBLICATION
Publication::~Publication(){
    delete this->author;

};

std::ostream& operator<<(std::ostream& os, const Publication& rhs) {
        char* p_name;
        switch(rhs.pub_type){
                case THESIS : p_name = new char[sizeof "THESIS"];
                        std::strcpy(p_name, "THESIS");
                        break;
                case ARTICLE : p_name = new char[sizeof "ARTICLE"];
                        std::strcpy(p_name, "ARTICLE");
```

```
                    break;
                case REPORT : p_name = new char[sizeof "REPORT"];
                    std::strcpy(p_name, "REPORT");
                    break;
                case INTARTICLE : p_name = new char[sizeof "INTARTICLE"];
                    std::strcpy(p_name, "INTARTICLE");
                    break;
        };
         os << *(rhs.author) << "Type: " << p_name << std::endl;
         return os;
}

std::istream& operator>>(std::istream& is, Publication& rhs) {
         is.clear();
        is.ignore(80, '\n');
        Student* author_;
        author_ = new Student();
        std::cin>>*author_;
        rhs.modAuthor(*author_);
        delete author_;
        char* ptype = new char[10];
        std::cout << "Input publication type (THESIS, ARTICLE, REPORT,
INTARTICLE):\n";
        is >> ptype;
        SciAchivment pub_type_;
        if (strcmp(ptype, "THESIS")==0) pub_type_=THESIS;
        if (strcmp(ptype, "ARTICLE")==0) pub_type_=ARTICLE;
        if (strcmp(ptype, "REPORT")==0) pub_type_=REPORT;
        if (strcmp(ptype, "INTARTICLE")==0)pub_type_=INTARTICLE;
        if (strcmp(ptype, "INTARTICLE")!=0 && strcmp(ptype, "REPORT")!=0 &&
strcmp(ptype, "ARTICLE")!=0 &&
                strcmp(ptype, "THESIS")!=0) pub_type_=THESIS;
        rhs.modPublicationType(pub_type_);
         return is;
}

Publication& Publication::operator=(const Publication& rhs){
        if (this == &rhs) return *this; // Gracefully handle self assignment
        delete author;
        author = new Student();
        *author = *rhs.author;
        pub_type=rhs.pub_type;
        return *this;
}
//-----------------------------------------------

//-----------------------------------------------

//CONSTRUCTORS RESEARCH

Research::Research(){
    this->sign_date = new Date();
    this->num_of_publications = 0;
    this->publications = NULL;

};

Research::Research(const Customer& in_customer, const Date&
in_date):Customer(in_customer){
    this->sign_date = new Date(in_date);
    this->num_of_publications = 0;
    this->publications = NULL;

};

Research::Research(const Research& in_research){
    strcpy(this->name, in_research.name);
    strcpy(this->theme, in_research.theme);
    this->price = in_research.price;
    this->sign_date = new Date(in_research.getSignDate());
    this->num_of_publications = in_research.getNumOfPublications();
    this->publications = new Publication*[this->num_of_publications];
    const Publication * const * retireved = in_research.getPublicationList();
    for (int i=0; i++; i < this->num_of_publications){
        *(this->publications + i) = new Publication(**(retireved + i));
    };

};

//CONSTRUCTORS RESEARCH


const Date& Research::getSignDate() const{
    return *(this->sign_date);
};

int Research::getNumOfPublications() const{
    return this->num_of_publications;
};

const Publication* const* Research::getPublicationList() const{
    return this->publications;
};

char* Research::getInfo() const{
    char* res = new char[(std::strlen(this->theme) + 1) +
                       sizeof "theme: \nnum of publications:" +
                       (sizeof (char))*(int)(std::log(this->num_of_publications ?
                                                this->num_of_publications != 0 :
                                                1)
                                          / std::log(10))];
    sprintf(res, "theme: %s\nnum of publications: %i",
                this->theme, this->num_of_publications);
    return res;
```

```
};

char* Research::out() const{
    //getting fields out-strings
    char* customer_out = Customer::out(); //using the base function out()
    char* date_out = this->sign_date->out();
    char* nop_out = new char[sizeof "\nNumber of all publications" +
                          (int)(1 + std::log((this->num_of_publications != 0 )? this-
>num_of_publications : 1) /
                                std::log(10))
                          ];
    //Formed number of all publications
    sprintf(nop_out, "%s:\t%i",
                "\nNumber of all publications", this->num_of_publications);
    //getting data from all publications and calulating their overall size
    char ** pubs_out = new char*[num_of_publications];
    int totalsize = 0;
    for (int i=0; i < this->num_of_publications; i++){
        *(pubs_out + i) = (*(this->publications + i))->out();
        totalsize += std::strlen(*(pubs_out + i));
    };
    //allocating resulting string, with size as sum of all pieces
    char * res = new char[sizeof "Customer information:\t" +
(std::strlen(customer_out) + 1) +
                          sizeof "Signing date:\t" + (std::strlen(date_out) + 1) +
                          (std::strlen(nop_out) + 1) +
                          sizeof "\nList of all publications:\n" + totalsize +
                          (sizeof "\t")*num_of_publications
                          ];
    //Collecting all strings in resulting string
    std::strcpy(res, "Customer information:\t");
    std::strcat(res, customer_out);
    std::strcat(res, "\nSigning date:\t");
    std::strcat(res, date_out);
    std::strcat(res, nop_out);
    std::strcat(res, "\nList of all publications:\n");
    for (int i=0; i < num_of_publications; i++){
        std::strcat(res, "\t");
        std::strcat(res, *(pubs_out + i));
    };
    //now it's time to deallocate these arrays
    delete[] customer_out;
    delete[] date_out;
    delete[] nop_out;
    for (int i=0; i < num_of_publications; i++){
        delete[] *(pubs_out + i);
    };
    delete [] pubs_out;
    return res;
};

Research& Research::addPublication(const Publication& in_publication){
    this->num_of_publications += 1;
    this->publications = (Publication**)std::realloc(this->publications, this-
>num_of_publications * sizeof (Publication*));
    *(this->publications + this->num_of_publications - 1) = new
Publication(in_publication);
    return *this;
};

Research& Research::modCustomer(const Customer& in_customer){
    this->modName(in_customer.getName());
    this->modTheme(in_customer.getTheme());
    this->modPrice(in_customer.getPrice());
    return *this;
};

bool Research::operator<(const Research& rhs){return (this->price <
rhs.price);}
bool Research::operator>(const Research& rhs){return (this->price >
rhs.price);}



char const* Research::operator[](int nSubscript){
        if( (nSubscript+1) > 0 && (nSubscript+1) <= this-
>getNumOfPublications() ){
                return (*(this->publications+nSubscript))->out();
        }
        else {
                std::clog << "Array bounds violation." <<std::endl;
                return "";
        }
}

Research& Research::operator=(const Research& rhs){
        if (this == &rhs) return *this; // Gracefully handle self assignment

        publications = (Publication**)std::realloc(publications,
rhs.num_of_publications * sizeof (Publication*));
        for (int i=num_of_publications; i< rhs.num_of_publications; i++){
                *(publications + i)=new Publication;
        };
        for (int i=0; i< rhs.num_of_publications; i++){
                (*(publications + i))->modAuthor((*(rhs.publications + i))-
>getAuthor());
                (*(publications + i))->modPublicationType((*(rhs.publications + i))-
>getPublicationType());
        };
        num_of_publications=rhs.num_of_publications;
        *sign_date = *rhs.sign_date;
        modName(rhs.getName());
        modTheme(rhs.getTheme());
        modPrice(rhs.getPrice());
        return *this;
}


//DESTRUCTOR RESEARCH
Research::~Research(){
    for (int i=0; i < num_of_publications; i++){
        delete *(this->publications + i);
    };
    std::free(this->publications);

};
```

lab1.cpp

```
#include "lab1mod.hpp"
#include <iostream>
#undef max
#include <limits>


int main(){
        Date x = Date(11, 11, 1596), y = Date(1,1,1490), z;
        z = (x + y);
        std::cout<<z.out()<<std::endl;

        std::cout<<"--------------------------------------------------"<<std::endl;
        std::cout<<"--------------------------------------------------"<<std::endl;
        Customer cust ("Andrew", "Lol", 1500), cust2;
        Research r1, r2 = Research(cust, z);
        std::cout<<(r1<r2)<<std::endl;
        std::cout<<"--------------------------"<<std::endl;
        std::cout<<(r1>r2)<<std::endl;

        std::cout<<"--------------------------------------------------"<<std::endl;
        std::cout<<"--------------------------------------------------"<<std::endl;
        std::cout<<r2[0]<<std::endl;
        std::cout<<"--------------------------"<<std::endl;
        Student stud1 ("Alex", "Trump", 2014), stud2 ("Donald", "Trump", 2014);
        Publication pub (stud1, ARTICLE), pub2;
        r2.addPublication(pub);
        std::cout<<r2[0]<<std::endl;

        std::cout<<"--------------------------------------------------"<<std::endl;
        std::cout<<"--------------------------------------------------"<<std::endl;
        std::cout<<(stud1==stud2)<<" "<<(stud1!=stud2)<<std::endl;
        stud2.modName("Alex");
        std::cout<<"--------------------------"<<std::endl;
        std::cout<<(stud1==stud2)<<" "<<(stud1!=stud2)<<std::endl;

        std::cout<<"--------------------------------------------------"<<std::endl;
```

```
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<stud1;
std::cout<<"----------------------------"<<std::endl;
std::cin>>stud2;
std::cout<<"----------------------------"<<std::endl;
std::cout<<stud2;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<pub;
std::cout<<"----------------------------"<<std::endl;
std::cin>>pub;
std::cout<<"----------------------------"<<std::endl;
std::cout<<pub;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<stud1<<std::endl;
stud1=stud2;
std::cout<<stud1<<std::endl;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<cust2.out()<<std::endl;
cust2=cust;
std::cout<<cust2.out()<<std::endl;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<pub2<<std::endl;
pub2=pub;
std::cout<<pub2<<std::endl;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<r2.out()<<std::endl;
std::cout<<"----------------------------"<<std::endl;
std::cout<<r1.out()<<std::endl;
std::cout<<"----------------------------"<<std::endl;
r1=r2;
std::cout<<r1.out()<<std::endl;

std::cout<<"------------------------------------------------------"<<std::endl;
std::cout<<"------------------------------------------------------"<<std::endl;
//реализуем полиморфизм
Customer custom_virt;
Research research_virt;
Customer* cust_v1 = &custom_virt;
Customer* cust_v2 = &research_virt;
std::cout<<cust_v1->out()<<std::endl;
std::cout<<"----------------------------"<<std::endl;
std::cout<<cust_v2->out()<<std::endl;
```

```
        return 0;
};
```

Результати роботи програми

```
alextr@alextr:/media/alextr/DATA/5 semester/OOP (C++,
C#)/Labwork3$ make lab1
g++ -c -std=c++11 -o obj/lab1.o lab1.cpp
g++ -c -std=c++11 -o obj/lab1mod.o lab1mod.cpp
g++ -o lab1 obj/lab1.o obj/lab1mod.o lab1mod.hpp
alextr@alextr:/media/alextr/DATA/5 semester/OOP (C++,
C#)/Labwork3$ ./lab1
12.12.3086
--------------------------------------------------
--------------------------------------------------
1
--------------------------
0
--------------------------------------------------
--------------------------------------------------
Array bounds violation.

--------------------------
Alex Trump
2014 Publication type : article in proffesional journal
--------------------------------------------------
--------------------------------------------------
0 1
--------------------------
1 0
--------------------------------------------------
--------------------------------------------------
==Name==Alex/==Surname== Trump/Enroll year: 2014
--------------------------

Input name:
Father
Input surname:
God
Input enroll year: 2009
--------------------------
==Name==Father/==Surname== God/Enroll year: 2009
--------------------------------------------------
--------------------------------------------------
==Name==Alex/==Surname== Trump/Enroll year: 2014
Type: ARTICLE
--------------------------

Input name:
Rooney
```

```
Input surname:
Mickky
Input enroll year: 1997
Input publication type (THESIS, ARTICLE, REPORT, INTARTICLE):
ARTICLE
---------------------------
==Name==Rooney/==Surname== Mickky/Enroll year: 1997
Type: ARTICLE
---------------------------------------------------
---------------------------------------------------
==Name==Alex/==Surname== Trump/Enroll year: 2014

==Name==Father/==Surname== God/Enroll year: 2009

---------------------------------------------------
---------------------------------------------------
Name Theme
0000
Andrew Lol
1500
---------------------------------------------------
---------------------------------------------------
==Name==Name/==Surname== Surname/Enroll year: 2016
Type: THESIS

==Name==Rooney/==Surname== Mickky/Enroll year: 1997
Type: ARTICLE

---------------------------------------------------
---------------------------------------------------
Customer information:  Andrew Lol
1500
Signing date:       12.12.3086
Number of all publications:    1
List of all publications:
        Alex Trump
2014 Publication type : article in proffesional journal
---------------------------
Customer information:  Name Theme
0000
Signing date:       16.11.2016
Number of all publications:    0
List of all publications:

---------------------------
Customer information:  Andrew Lol
1500
Signing date:       12.12.3086
Number of all publications:    1
List of all publications:
        Alex Trump
2014 Publication type : article in proffesional journal
```


Name Theme
0000

Customer information: Name Theme
0000
Signing date: 16.11.2016
Number of all publications: 0
List of all publications: