Навчально-науковий комплекс “Інститут прикладного системного аналізу”

при Національному технічному університеті України “КПІ”

Кафедра математичних методів системного аналізу

РОБОТИ КОМП’ЮТЕРНОГО ПРАКТИКУМУ

З КУРСУ “ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ”

Виконав: студент 2-го курсу

групи КА-04

Штріккер Д.Я.

Прийняв: Шолохов О.В.

**Мета роботи:** Навчитись коректно користуватися перевантаженням функцій,   
перевантаженням операцій та механізмом віртуальних функцій в С++.

**Завдання (варіант 8):**

* 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.

**Лістинг програми:**

Github (разом із історією створення): <https://github.com/dindin28/task_3>

**Класс “Date”**

**Header:**

#ifndef \_TASK\_3\_INCLUDE\_TASK\_3\_DATE\_H\_

#define \_TASK\_3\_INCLUDE\_TASK\_3\_DATE\_H\_

#include <iostream>

class Date

{

public:

Date();

Date(int day, int month, int year);

Date(const Date &copy);

~Date();

//Getters

int GetDay() const;

int GetMonth() const;

int GetYear() const;

//Selectors

Date &SetDay(int day);

Date &SetMonth(int month);

Date &SetYear(int year);

//Helper

bool IsYearLeap();

static bool IsYearLeap(int year);

virtual void Print();

friend std::ostream &operator<<(std::ostream &out, const Date &obj);

friend std::istream &operator>>(std::istream &in, Date &obj);

Date operator+(const Date &obj);

bool operator==(const Date &obj);

bool operator!=(const Date &obj);

private:

int day\_, month\_, year\_;

}; //Class (Date)

#endif //Header Guard

**Source file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_3/date.h>

#include <ctime>

#include <iomanip>

#include <cstdlib>

#include <iostream>

Date::Date()

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Date()" << std::endl;

#endif

std::time\_t now = std::time(nullptr);

char buff[4];

strftime(buff, sizeof(buff) + 1, "%d", std::localtime(&now));

day\_ = atoi(buff);

strftime(buff, sizeof(buff) + 1, "%m", std::localtime(&now));

month\_ = atoi(buff);

strftime(buff, sizeof(buff) + 1, "%Y", std::localtime(&now));

year\_ = atoi(buff);

}

Date::Date(int day, int month, int year)

: day\_(day), month\_(month), year\_(year)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Date(int day, int month, int year)" << std::endl;

#endif

}

Date::Date(const Date &copy)

: day\_(copy.day\_), month\_(copy.month\_), year\_(copy.year\_)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Date(const Date &copy)" << std::endl;

#endif

}

Date::~Date()

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "~Date()" << std::endl;

#endif

}

//Getters

int Date::GetDay() const { return day\_; }

int Date::GetMonth() const { return month\_; }

int Date::GetYear() const { return year\_; }

//Selectors

Date &Date::SetDay(int day)

{

day\_ = day;

return \*this;

}

Date &Date::SetMonth(int month)

{

month\_ = month;

return \*this;

}

Date &Date::SetYear(int year)

{

year\_ = year;

return \*this;

}

void Date::Print()

{

std::cout << std::setfill('0');

std::cout << std::setw(2) << day\_ << "/"

<< std::setw(2) << month\_ << "/"

<< std::setw(4) << year\_ << std::endl;

}

Date Date::operator+(const Date &obj)

{

int \_day = obj.day\_ + day\_;

int \_month = obj.month\_ + month\_;

int \_year = obj.year\_ + year\_;

while (\_month > 12)

{

\_year++;

\_month -= 12;

}

int dayInYear = \_day;

\_month--;

while (\_month > 0)

{

if (\_month == 1 || \_month == 3 || \_month == 5 || \_month == 7 || \_month == 8 || \_month == 10 || \_month == 12)

{

\_month--;

dayInYear += 31;

continue;

}

if ((\_month == 2 && !IsYearLeap(\_year)) || (\_month == 2 && IsYearLeap(\_year)))

{

\_month--;

if (IsYearLeap(\_year))

{

dayInYear += 29;

}

else

{

dayInYear += 28;

}

continue;

}

if ((\_month == 4 || \_month == 6 || \_month == 9 || \_month == 11))

{

\_month--;

dayInYear += 30;

continue;

}

}

\_month = 1;

if ((dayInYear > 365 && !IsYearLeap(\_year)) || (dayInYear > 366 && IsYearLeap(\_year)))

{

\_year++;

if (IsYearLeap(\_year))

{

dayInYear -= 366;

}

else

{

dayInYear -= 365;

}

}

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //January

if ((dayInYear > 28 && !IsYearLeap(\_year)) || (dayInYear > 29 && IsYearLeap(\_year)))

{

\_month++;

if (IsYearLeap(\_year))

{

dayInYear -= 29;

}

else

{

dayInYear -= 28;

}

} //February

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //March

if (dayInYear > 30)

{

\_month++;

dayInYear -= 30;

} //April

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //May

if (dayInYear > 30)

{

\_month++;

dayInYear -= 30;

} //June

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //July

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //August

if (dayInYear > 30)

{

\_month++;

dayInYear -= 30;

} //September

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //October

if (dayInYear > 30)

{

\_month++;

dayInYear -= 30;

} //November

if (dayInYear > 31)

{

\_month++;

dayInYear -= 31;

} //December

return Date(dayInYear, \_month, \_year);

}

bool Date::IsYearLeap(int year)

{

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

return true;

return false;

} //Static Function (IsYearLeap)

bool Date::IsYearLeap()

{

if ((year\_ % 4 == 0 && year\_ % 100 != 0) || (year\_ % 400 == 0))

return true;

return false;

} //Function (IsYearLeap)

std::ostream &operator<<(std::ostream &out, const Date &obj)

{

out << std::setfill('0');

out << std::setw(2) << obj.day\_ << "/"

<< std::setw(2) << obj.month\_ << "/"

<< std::setw(4) << obj.year\_;

return out;

}

std::istream &operator>>(std::istream &in, Date &obj)

{

std::cout << "Enter new day of date: ";

in >> obj.day\_;

std::cout << "Enter new month of date: ";

in >> obj.month\_;

std::cout << "Enter new year of date: ";

in >> obj.year\_;

return in;

}

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

{

return (day\_ == obj.day\_ && month\_ == obj.month\_ && year\_ == obj.year\_);

}

bool Date::operator!=(const Date &obj)

{

return !Date::operator==(obj);

}

**Клас “Participant”**

**Header:**

**#ifndef \_TASK\_3\_INCLUDE\_TASK\_3\_PARTICIPANT\_H\_**

#define \_TASK\_3\_INCLUDE\_TASK\_3\_PARTICIPANT\_H\_

#include <iostream>

#include <task\_3/date.h>

class Participant

{

public:

Participant();

Participant(const char \*name, const char \*surname, const Date &date);

Participant(const Participant &copy);

Participant &operator=(const Participant &copy);

virtual ~Participant();

//Getters

const char \*GetName() const;

const char \*GetSurname() const;

Date GetDate() const;

//Selectors

Participant &SetName(const char \*name);

Participant &SetSurname(const char \*surname);

Participant &SetDate(const Date &date);

virtual void Print();

friend std::ostream &operator<<(std::ostream &out, const Participant &obj);

friend std::istream &operator>>(std::istream &in, Participant &obj);

bool operator==(const Participant &obj);

bool operator!=(const Participant &obj);

private:

char \*name\_, \*surname\_;

Date date\_;

}; //Class (Participant)

#endif //Header Guard

**Source file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_3/participant.h>

#include <cstring>

#include <string>

Participant::Participant()

: date\_(15, 8, 2002)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Participant()" << std::endl;

#endif

const char \*name = "Dima";

const char \*surname = "Shtrikker";

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

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

strcpy(name\_, name);

strcpy(surname\_, surname);

}

Participant::Participant(const char \*name, const char \*surname, const Date &date)

: date\_(date)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Participant(const char \*name, const char \*surname, const Date &date)" << std::endl;

#endif

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

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

strcpy(name\_, name);

strcpy(surname\_, surname);

}

Participant::Participant(const Participant &copy)

: date\_(copy.date\_)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Participant(const Participant &copy)" << std::endl;

#endif

name\_ = new char[strlen(copy.name\_) + 1];

surname\_ = new char[strlen(copy.surname\_) + 1];

strcpy(name\_, copy.name\_);

strcpy(surname\_, copy.surname\_);

}

Participant &Participant::operator=(const Participant &copy)

{

if (this != &copy)

{

//Name

if (strlen(name\_) != 0)

{

delete[] name\_;

}

name\_ = new char[strlen(copy.name\_) + 1];

strcpy(name\_, copy.name\_);

//Surname

if (strlen(surname\_) != 0)

{

delete[] surname\_;

}

surname\_ = new char[strlen(copy.surname\_) + 1];

strcpy(surname\_, copy.surname\_);

//Date

date\_ = copy.date\_;

} //Condition(this != &copy)

return \*this;

}

Participant::~Participant()

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "~Participant()" << std::endl;

#endif

if (strlen(name\_) != 0)

{

delete[] name\_;

}

if (strlen(surname\_) != 0)

{

delete[] surname\_;

}

}

//Getters

const char \*Participant::GetName() const { return name\_; }

const char \*Participant::GetSurname() const { return surname\_; }

Date Participant::GetDate() const { return date\_; }

//Selectors

Participant &Participant::SetName(const char \*name)

{

if (strlen(name\_) != 0)

{

delete[] name\_;

}

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

strcpy(name\_, name);

return \*this;

}

Participant &Participant::SetSurname(const char \*surname)

{

if (strlen(surname\_) != 0)

{

delete[] surname\_;

}

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

strcpy(surname\_, surname);

return \*this;

}

Participant &Participant::SetDate(const Date &date)

{

date\_ = date;

return \*this;

}

void Participant::Print()

{

std::cout << name\_ << " "

<< surname\_ << " ("

<< date\_ << ")" << std::endl;

}

std::ostream &operator<<(std::ostream &out, const Participant &obj)

{

out << obj.name\_ << " "

<< obj.surname\_ << " ("

<< obj.date\_ << ")";

return out;

}

std::istream &operator>>(std::istream &in, Participant &obj)

{

std::string buff\_string;

//Name

std::cout << "Enter new name of participant: ";

in >> buff\_string;

if (strlen(obj.name\_) != 0)

{

delete[] obj.name\_;

}

obj.name\_ = new char[buff\_string.length() + 1];

strcpy(obj.name\_, buff\_string.c\_str());

//Surname

std::cout << "Enter new surname of participant: ";

in >> buff\_string;

if (strlen(obj.surname\_) != 0)

{

delete[] obj.surname\_;

}

obj.surname\_ = new char[buff\_string.length() + 1];

strcpy(obj.surname\_, buff\_string.c\_str());

//Date

in >> obj.date\_;

return in;

}

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

{

return (strcmp(name\_, obj.name\_) == 0 &&

strcmp(surname\_, obj.surname\_) == 0 &&

date\_ == obj.date\_);

}

bool Participant::operator!=(const Participant &obj)

{

return !Participant::operator==(obj);

}

**Клас “Performance”**

**Header:**

#ifndef \_TASK\_3\_INCLUDE\_TASK\_3\_PERFORMANCE\_H\_

#define \_TASK\_3\_INCLUDE\_TASK\_3\_PERFORMANCE\_H\_

#include <task\_3/participant.h>

class Performance

{

public:

Performance();

Performance(bool performance\_type, Participant participant, int sequence\_number, int result);

Performance(const Performance &copy);

Performance &operator=(const Performance &copy);

~Performance();

//Getters

bool GetPerformanceType() const;

int GetNumberOfCompetitions() const;

Participant GetParticipant() const;

int GetSequenceNumber() const;

int GetResult() const;

//Selectors

Performance &SetPerformanceType(bool performance\_type);

Performance &SetParticipant(Participant participant);

Performance &SetSequenceNumber(int sequence\_number);

Performance &SetResult(int result);

virtual void Print();

friend std::ostream &operator<<(std::ostream &out, const Performance &obj);

friend std::istream &operator>>(std::istream &in, Performance &obj);

bool operator<(const Performance &obj);

bool operator>(const Performance &obj);

private:

// performance\_type\_= 0(false) => team performance

// performance\_type\_= 1(true) => individual performance

bool performance\_type\_;

static int number\_of\_performance\_;

Participant participant\_;

int sequence\_number\_, result\_;

}; //Class (Performance)

#endif //Header Guard

**Source file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_3/performance.h>

//Initialization of static variables

int Performance::number\_of\_performance\_ = 0;

// performance\_type\_= 0(false) => team performance

// performance\_type\_= 1(true) => individual performance

Performance::Performance()

: performance\_type\_(true),

participant\_(Participant()),

sequence\_number\_(1),

result\_(100)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Performance()" << std::endl;

#endif

number\_of\_performance\_++;

}

Performance::Performance(bool performance\_type, Participant participant, int sequence\_number, int result)

: performance\_type\_(performance\_type),

participant\_(participant),

sequence\_number\_(sequence\_number),

result\_(result)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Performance(bool performance\_type, Participant participant, int sequence\_number, int result)"

<< std::endl;

#endif

number\_of\_performance\_++;

}

Performance::Performance(const Performance &copy)

: performance\_type\_(copy.performance\_type\_),

participant\_(copy.participant\_),

sequence\_number\_(copy.sequence\_number\_),

result\_(copy.result\_)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Performance(const Performance &copy)" << std::endl;

#endif

number\_of\_performance\_++;

}

Performance &Performance::operator=(const Performance &copy)

{

performance\_type\_ = copy.performance\_type\_;

participant\_ = copy.participant\_;

sequence\_number\_ = copy.sequence\_number\_;

result\_ = copy.result\_;

return \*this;

}

Performance::~Performance()

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "~Performance()" << std::endl;

#endif

}

//Getters

bool Performance::GetPerformanceType() const { return performance\_type\_; }

int Performance::GetNumberOfCompetitions() const { return number\_of\_performance\_; }

Participant Performance::GetParticipant() const { return participant\_; }

int Performance::GetSequenceNumber() const { return sequence\_number\_; }

int Performance::GetResult() const { return result\_; }

//Selectors

Performance &Performance::SetPerformanceType(bool performance\_type)

{

performance\_type\_ = performance\_type;

return \*this;

}

Performance &Performance::SetParticipant(Participant participant)

{

participant\_ = participant;

return \*this;

}

Performance &Performance::SetSequenceNumber(int sequence\_number)

{

sequence\_number\_ = sequence\_number;

return \*this;

}

Performance &Performance::SetResult(int result)

{

result\_ = result;

return \*this;

}

void Performance::Print()

{

// performance\_type\_= 0(false) => team performance

// performance\_type\_= 1(true) => individual performance

if (performance\_type\_ == false)

{

std::cout << "Team performance. ";

}

else

{

std::cout << "Individual performance. ";

}

std::cout << participant\_ << "(" << sequence\_number\_ << ", " << result\_ << ")" << std::endl;

}

std::ostream &operator<<(std::ostream &out, const Performance &obj)

{

// performance\_type\_= 0(false) => team performance

// performance\_type\_= 1(true) => individual performance

if (obj.performance\_type\_ == false)

{

out << "Team performance. ";

}

else

{

out << "Individual performance. ";

}

out << obj.participant\_ << "(" << obj.sequence\_number\_ << ", " << obj.result\_ << ")";

return out;

}

std::istream &operator>>(std::istream &in, Performance &obj)

{

//Performance\_type

std::cout << "Enter new performance type (1 - individual; 0 - team): ";

in >> obj.performance\_type\_;

//Participant

in >> obj.participant\_;

//Sequence number

std::cout << "Enter new sequence number for participant: ";

in >> obj.sequence\_number\_;

//Result

std::cout << "Enter new result for participant: ";

in >> obj.result\_;

return in;

}

bool Performance::operator>(const Performance &obj)

{

return (this->result\_ > obj.result\_);

}

bool Performance::operator<(const Performance &obj)

{

return (this->result\_ < obj.result\_);

}

**Клас “Competition”**

**Header:**

#ifndef \_TASK\_3\_INCLUDE\_TASK\_3\_COMPETITION\_H\_

#define \_TASK\_3\_INCLUDE\_TASK\_3\_COMPETITION\_H\_

#include <task\_3/performance.h>

#include <task\_3/participant.h>

class Competition : public Participant

{

public:

Competition();

Competition(const char \*name);

Competition(const char \*name, Performance \*performance\_pointer, int performance\_size);

Competition(const char \*name, Performance \*performance\_pointer, int performance\_size, const char \*par\_name, const char \*par\_surname, const Date &par\_date);

Competition(const Competition &copy);

Competition &operator=(const Competition &copy);

virtual ~Competition();

void AddPerformance(const Performance &performance);

const char \*GetCompetitionName() const;

Competition &SetCompetitionName(const char \*name);

virtual void Print();

virtual void PrintShortly();

friend std::ostream &operator<<(std::ostream &out, const Competition &obj);

friend std::istream &operator>>(std::istream &in, Competition &obj);

Performance &operator[](const int index);

private:

char \*competition\_name\_;

Performance \*performance\_pointer\_;

int performance\_size\_;

}; //Class (Competition)

#endif //Header Guard

**Source file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_3/competition.h>

#include <cstring>

#include <string>

Competition::Competition()

: performance\_size\_(0), performance\_pointer\_(nullptr), Participant(Participant())

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Competition()" << std::endl;

#endif

const char \*name = "Olympic Games";

competition\_name\_ = new char[strlen(name) + 1];

strcpy(competition\_name\_, name);

}

Competition::Competition(const char \*name)

: performance\_size\_(0), performance\_pointer\_(nullptr), Participant(Participant())

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Competition(const char \*name)" << std::endl;

#endif

competition\_name\_ = new char[strlen(name) + 1];

strcpy(competition\_name\_, name);

}

Competition::Competition(const char \*name, Performance \*performance\_pointer, int performance\_size)

: performance\_size\_(performance\_size), Participant(Participant())

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Competition(const char \*name, Performance \*performance\_pointer, int performance\_size)" << std::endl;

#endif

//copy name

competition\_name\_ = new char[strlen(name) + 1];

strcpy(competition\_name\_, name);

//copy performance\_pointer

performance\_pointer\_ = new Performance[performance\_size];

for (int i = 0; i < performance\_size; i++)

{

performance\_pointer\_[i] = performance\_pointer[i];

}

}

Competition::Competition(const char \*name, Performance \*performance\_pointer, int performance\_size, const char \*par\_name, const char \*par\_surname, const Date &par\_date)

: performance\_size\_(performance\_size), Participant(par\_name, par\_surname, par\_date)

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Competition(const char \*name, Performance \*performance\_pointer, int performance\_size, const char \*par\_name, const char \*par\_surname, const Date &par\_date)"

<< std::endl;

#endif

//copy name

competition\_name\_ = new char[strlen(name) + 1];

strcpy(competition\_name\_, name);

//copy performance\_pointer

performance\_pointer\_ = new Performance[performance\_size];

for (int i = 0; i < performance\_size; i++)

{

performance\_pointer\_[i] = performance\_pointer[i];

}

}

Competition::Competition(const Competition &copy)

: performance\_size\_(copy.performance\_size\_), Participant(copy.GetName(), copy.GetSurname(), copy.GetDate())

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "Competition(const Competition &copy)" << std::endl;

#endif

//copy name

competition\_name\_ = new char[strlen(copy.competition\_name\_) + 1];

strcpy(competition\_name\_, copy.competition\_name\_);

//copy performance\_pointer

performance\_pointer\_ = new Performance[copy.performance\_size\_];

for (int i = 0; i < copy.performance\_size\_; i++)

{

performance\_pointer\_[i] = copy.performance\_pointer\_[i];

}

}

Competition &Competition::operator=(const Competition &copy)

{

if (this != &copy)

{

//Participant variables

SetName(copy.GetName());

SetSurname(copy.GetSurname());

SetDate(copy.GetDate());

//Name

if (strlen(competition\_name\_) != 0)

{

delete[] competition\_name\_;

}

competition\_name\_ = new char[strlen(copy.competition\_name\_) + 1];

strcpy(competition\_name\_, copy.competition\_name\_);

//Performance pointer

if (performance\_size\_ != 0)

{

delete[] performance\_pointer\_;

}

performance\_pointer\_ = new Performance[copy.performance\_size\_];

for (int i = 0; i < copy.performance\_size\_; i++)

{

performance\_pointer\_[i] = copy.performance\_pointer\_[i];

}

//Performance size

performance\_size\_ = copy.performance\_size\_;

} //Condition (this != &copy)

return \*this;

}

Competition::~Competition()

{

#ifdef SHOW\_CONSTRUCTORS

std::cout << "~Competition()" << std::endl;

#endif

if (strlen(competition\_name\_) != 0)

{

delete[] competition\_name\_;

}

if (performance\_size\_ != 0)

{

delete[] performance\_pointer\_;

}

}

void Competition::AddPerformance(const Performance &performance)

{

Performance \*new\_ptr = new Performance[performance\_size\_ + 1];

for (int i = 0; i < performance\_size\_; i++)

{

new\_ptr[i] = performance\_pointer\_[i];

}

new\_ptr[performance\_size\_] = performance;

if (performance\_size\_ != 0)

{

delete[] performance\_pointer\_;

}

performance\_pointer\_ = new\_ptr;

performance\_size\_++;

}

const char \*Competition::GetCompetitionName() const { return competition\_name\_; }

Competition &Competition::SetCompetitionName(const char \*name)

{

if (strlen(competition\_name\_) != 0)

{

delete[] competition\_name\_;

}

competition\_name\_ = new char[strlen(name) + 1];

strcpy(competition\_name\_, name);

return \*this;

}

void Competition::Print()

{

Participant::Print();

std::cout << competition\_name\_ << std::endl;

for (int i = 0; i < performance\_size\_; i++)

{

std::cout << i + 1 << ". " << performance\_pointer\_[i] << std::endl;

}

}

void Competition::PrintShortly()

{

Participant::Print();

std::cout << competition\_name\_ << std::endl;

for (int i = 0; i < performance\_size\_; i++)

{

std::cout << i + 1 << ". " << performance\_pointer\_[i].GetParticipant().GetSurname() << std::endl;

}

}

std::ostream &operator<<(std::ostream &out, const Competition &obj)

{

//Participant out

out << obj.GetName() << " "

<< obj.GetSurname() << " ("

<< obj.GetDate() << ")" << std::endl;

//Competition out

out << obj.competition\_name\_ << std::endl;

for (int i = 0; i < obj.performance\_size\_; i++)

{

out << i + 1 << ") " << obj.performance\_pointer\_[i] << std::endl;

}

return out;

}

std::istream &operator>>(std::istream &in, Competition &obj)

{

std::string buff\_string;

std::cout << "Enter new name of competition: ";

in >> buff\_string;

if (strlen(obj.competition\_name\_) != 0)

{

delete[] obj.competition\_name\_;

}

obj.competition\_name\_ = new char[buff\_string.length() + 1];

strcpy(obj.competition\_name\_, buff\_string.c\_str());

return in;

}

Performance &Competition::operator[](const int index)

{

return performance\_pointer\_[index];

}

**Main file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_3/date.h>

#include <task\_3/participant.h>

#include <task\_3/performance.h>

#include <task\_3/competition.h>

#include <iostream>

int main()

{

Date date\_a(28, 2, 2020);

Date date\_b(1, 0, 0);

std::cout << "Year is leap, has 29 of Fabruary: " << date\_a << std::endl;

std::cout << "Just date with one day: " << date\_b << std::endl;

std::cout << "Sum is " << (date\_a + date\_b) << std::endl;

std::cout << std::endl;

date\_a.SetYear(2021);

std::cout << "Set first date with no leap year: " << date\_a << std::endl;

std::cout << "Now sum is " << (date\_a + date\_b) << std::endl;

std::cout << std::endl;

Performance per\_a;

Performance per\_b;

std::cout << "There is no diffrence between two performances: " << std::endl;

std::cout << "First performance: " << per\_a << std::endl;

std::cout << "Second performance: " << per\_b << std::endl;

std::cout << "per\_a > per\_b: " << (per\_a > per\_b) << std::endl;

std::cout << "per\_a < per\_b: " << (per\_a > per\_b) << std::endl; // Compared by result

std::cout << std::endl;

Participant par\_a;

Participant par\_b;

par\_a.SetName("Oleh");

std::cout << "Diffrence between two participants is name: " << std::endl;

std::cout << "First participant: " << par\_a << std::endl;

std::cout << "Second participant: " << par\_b << std::endl;

std::cout << "par\_a == par\_b: " << (par\_a == par\_b) << std::endl;

std::cout << "par\_a != par\_b: " << (par\_a != par\_b) << std::endl;

std::cout << std::endl;

Competition competition;

competition.AddPerformance(per\_a);

per\_a.SetResult(101);

competition.AddPerformance(per\_a);

per\_a.SetResult(102);

competition.AddPerformance(per\_a);

std::cout << "There are 3 diffrent performances in Competition: " << std::endl;

std::cout << competition << std::endl;

std::cout << "competition[1] = " << competition[1] << std::endl;

std::cout << std::endl;

Competition com\_ptr;

Participant \*a = &com\_ptr;

std::cout << "Participant class is parent for Competition class" << std::endl;

std::cout << "We have virtual Print() function: " << std::endl;

std::cout << "Participant a\* = &(Competition class), a->Print(): " << std::endl;

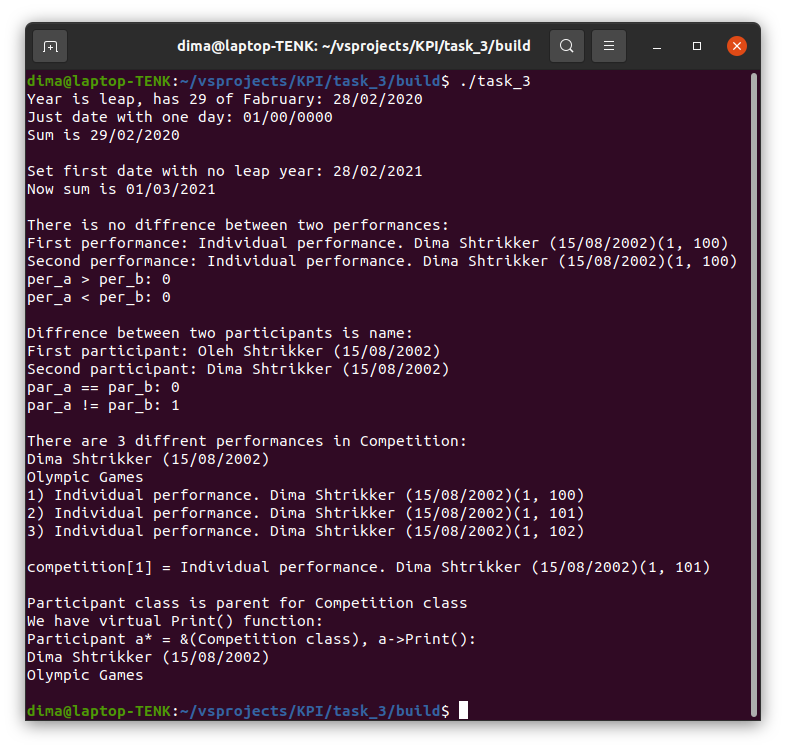
a->Print();

std::cout << std::endl;

return 0;

}

**Результати роботи програми:**

****

**Висновок:** Навчилися коректно користуватися перевантаженням функцій,   
перевантаженням операцій та механізмом віртуальних функцій в С++. Додатково перевантажили оператор == класу Date для зручності перевантаження оператора == класу Participant. Перевантажили оператори уведення-виведення для всіх класів. Та зробили функції виводу віртуальними.