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

Олексюк М.І.

Прийняв: Шолохов О.В.

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

**Завдання (варіант 18):**

Відповідно варіанту до тексту другої лабораторної роботи потрібно внести наступні зміни:

* арифметичний оператор «+» для класу «Пиломатеріал» за умови збігу полів типу «Деревина»; реалізувати як об’єднання;
* логічні оператори «<»та «>» для класу «Пиломатеріал» за вартістю;
* - логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Деревина»;
* - оператор індексації для доступу до інформації про пиломатеріал – для класу «Майстерня»;
* - оператори форматного уведення-виведення – для класів «Деревина» та «Пиломатеріал».

**Лістинг програми:**

**Класс “Date”**

**Header:**

#ifndef \_KP3\_INCLUDE\_KP3\_DATE\_H\_

#define \_KP3\_INCLUDE\_KP3\_DATE\_H\_

#include <iostream>

class Date

{

public:

Date();

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

Date(const Date &copy);

~Date();

// Getters

int GetDay();

int GetMonth();

int GetYear();

// Selectors

Date &SetDay(int day);

Date &SetMonth(int month);

Date &SetYear(int year);

// Extra functions

void Print();

static bool IsYearLeap(int year);

// Operators

bool operator==(const Date &copy);

bool operator!=(const Date &copy);

Date operator+(const Date &obj);

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

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

private:

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

}; // Class (Date)

#endif // Header Guard

**Source file:**

#include <KP3/date.h>

#include <ctime>

#include <iomanip>

#include <cstdlib>

#include <iostream>

Date::Date()

{

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);

/\*

time\_t timenow = time(0); //сьогоднішня дата

tm\* now = localtime(&timenow);

day = now->tm\_mday;

month = (now->tm\_mon + 1);

year = (now->tm\_year + 1900);

\*/

}

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

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

{}

Date::Date(const Date &copy)

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

{}

Date::~Date()

{}

// Getters

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

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

int Date::GetYear() { 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;

}

// Operators

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

{

return (day\_ == copy.day\_ &&

month\_ == copy.month\_ &&

year\_ == copy.year\_);

}

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

{

return !operator==(copy);

}

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;

} // 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;

}

**Enum “SawingOption”**

**Header:**

#ifndef \_KP3\_INCLUDE\_KP3\_SAWINGOPTION\_H\_

#define \_KP3\_INCLUDE\_KP3\_SAWINGOPTION\_H\_

#include <string>

enum SawingOption

{

kBeam = 1,

kUnedgedBoard = 2,

kEdgedBoard = 3,

kRail = 4,

};

std::string SawingOptionToString(SawingOption option);

SawingOption IntToSawingOption(int option);

#endif // Header Guard

**Source file:**

#include <KP3/sawing\_option.h>

std::string SawingOptionToString(SawingOption option)

{

std::string return\_str;

switch (option)

{

case (kBeam):

{

return\_str = "Beam";

break;

}

case (kUnedgedBoard):

{

return\_str = "Unedged Board";

break;

}

case (kEdgedBoard):

{

return\_str = "Edged Board";

break;

}

case (kRail):

{

return\_str = "Rail";

break;

}

}

return return\_str;

} // Function (SawingOptionToString)

SawingOption IntToSawingOption(int option)

{

SawingOption return\_option {};

switch (option)

{

case (kBeam):

{

return\_option = kBeam;

break;

}

case (kUnedgedBoard):

{

return\_option = kUnedgedBoard;

break;

}

case (kEdgedBoard):

{

return\_option = kEdgedBoard;

break;

}

case (kRail):

{

return\_option = kRail;

break;

}

}

return return\_option;

} // Function (IntToSawingOption)

**Клас “Wood”**

**Header:**

#ifndef \_KP3\_INCLUDE\_KP3\_WOOD\_H\_

#define \_KP3\_INCLUDE\_KP3\_WOOD\_H\_

#include <iostream>

class Wood

{

public:

// Default constructor(with no parameters)

Wood();

// Constructor with parameters

Wood(const char \*tree\_species, int wood\_moisture, int wood\_density);

// Copy constructor

Wood(const Wood &copy);

// operator =

Wood &operator=(const Wood &copy);

// Destructor

~Wood();

// Setters

Wood &SetTreeSpecies(const char \*tree\_species);

Wood &SetWoodMoisture(int wood\_moisture);

Wood &SetWoodDensity(int wood\_density);

// Getters

const char \*GetTreeSpecies() const;

int GetWoodMoisture() const;

int GetWoodDensity() const;

// Function to print class info

virtual void Print();

// Operators

bool operator==(const Wood &copy);

bool operator!=(const Wood &copy);

friend std::ostream &operator<<(std::ostream &out, const Wood &copy);

friend std::istream &operator>>(std::istream &in, Wood &copy);

private:

char \*tree\_species\_;

int wood\_moisture\_, wood\_density\_;

}; // Class(Wood)

#endif // Header Guard

**Source file:**

#include <KP3/wood.h>

#include <cstring>

#include <iostream>

Wood::Wood()

: wood\_moisture\_(0), wood\_density\_(80)

{

const char \*tree\_species = "Pine";

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

strcpy(tree\_species\_, tree\_species);

}

// Constructor with parameters

Wood::Wood(const char \*tree\_species, int wood\_moisture, int wood\_density)

: wood\_moisture\_(wood\_moisture), wood\_density\_(wood\_density)

{

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

strcpy(tree\_species\_, tree\_species);

}

// Copy constructor

Wood::Wood(const Wood &copy)

: wood\_moisture\_(copy.wood\_moisture\_), wood\_density\_(copy.wood\_density\_)

{

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

strcpy(tree\_species\_, copy.tree\_species\_);

}

// operator =

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

{

if (this != &copy) // Protect from leak of memory when this == &copy

{

// Copy tree\_species\_

if (strlen(tree\_species\_) != 0)

{

delete[] tree\_species\_;

}

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

strcpy(tree\_species\_, copy.tree\_species\_);

// Copy wood\_moisture\_

wood\_moisture\_ = copy.wood\_moisture\_;

// Copy wood\_density\_

wood\_density\_ = copy.wood\_density\_;

}

return \*this;

}

// Destructor

Wood::~Wood()

{

if (strlen(tree\_species\_) != 0)

{

delete[] tree\_species\_;

}

}

// Setters

Wood &Wood::SetTreeSpecies(const char \*tree\_species)

{

if (strlen(tree\_species\_) != 0)

{

delete[] tree\_species\_;

}

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

strcpy(tree\_species\_, tree\_species);

return \*this;

}

Wood &Wood::SetWoodMoisture(int wood\_moisture)

{

wood\_moisture\_ = wood\_moisture;

return \*this;

}

Wood &Wood::SetWoodDensity(int wood\_density)

{

wood\_density\_ = wood\_density;

return \*this;

}

// Getters

const char \*Wood::GetTreeSpecies() const { return tree\_species\_; }

int Wood::GetWoodMoisture() const { return wood\_moisture\_; }

int Wood::GetWoodDensity() const { return wood\_density\_; }

// Function to print class info

void Wood::Print()

{

std::cout << tree\_species\_ << "(" << wood\_moisture\_ << ", " << wood\_density\_ << ")" << std::endl;

}

// Operators

bool Wood::operator==(const Wood &copy)

{

return (strcmp(tree\_species\_, copy.tree\_species\_) == 0 &&

wood\_density\_ == copy.wood\_density\_ &&

wood\_moisture\_ == copy.wood\_moisture\_);

}

bool Wood::operator!=(const Wood &copy)

{

return !operator==(copy);

}

std::ostream &operator<<(std::ostream &out, const Wood &copy)

{

out << copy.GetTreeSpecies() << "(" << copy.GetWoodMoisture() << ", " << copy.GetWoodDensity() << ")";

return out;

}

std::istream &operator>>(std::istream &in, Wood &copy)

{

std::string buff\_str;

std::cout << "Enter Tree Species: ";

in >> buff\_str;

copy.SetTreeSpecies(buff\_str.c\_str());

int buff\_int;

std::cout << "Enter Wood Moisture: ";

in >> buff\_int;

copy.SetWoodMoisture(buff\_int);

std::cout << "Enter Wood Density: ";

in >> buff\_int;

copy.SetWoodDensity(buff\_int);

return in;

}

**Клас “Lumber”**

**Header:**

#ifndef \_KP3\_INCLUDE\_KP3\_LUMBER\_H\_

#define \_KP3\_INCLUDE\_KP3\_LUMBER\_H\_

#include <KP3/wood.h>

#include <KP3/date.h>

#include <KP3/sawing\_option.h>

class Lumber : public Wood

{

public:

// Default constructor(with no parameters)

Lumber();

// Constructor with parameters

Lumber(Wood wood,

SawingOption sawing\_option,

Date delivery\_date,

int marking, int amount, int cost);

// Copy constructor

Lumber(const Lumber &copy);

// Destructor

~Lumber();

// Setters

Lumber &SetWood(const Wood &wood);

Lumber &SetSawingOption(const SawingOption sawing\_option);

Lumber &SetDeliveryDate(const Date &delivery\_date);

Lumber &SetMarking(const int marking);

Lumber &SetAmount(const int amount);

Lumber &SetCost(const int cost);

// Getters

Wood GetWood() const;

SawingOption GetSawingOption() const;

Date GetDeliveryDate() const;

int GetMarking() const;

int GetAmount() const;

int GetCost() const;

// Function for printing class info

virtual void Print();

// Operators

Lumber operator+(const Lumber& obj);

bool operator<(const Lumber& obj);

bool operator>(const Lumber& obj);

friend std::ostream &operator<<(std::ostream &out, const Lumber &copy);

friend std::istream &operator>>(std::istream &in, Lumber &copy);

private:

SawingOption sawing\_option\_;

Date delivery\_date\_;

int marking\_, amount\_, cost\_;

}; // Class (Lumber)

#endif // Header Guard

**Source file:**

#include <KP3/lumber.h>

#include <iostream>

// Default constructor(with no parameters)

Lumber::Lumber()

: Wood(), sawing\_option\_(kBeam), delivery\_date\_(Date()), marking\_(0), amount\_(5), cost\_(100)

{}

// Constructor with parameters

Lumber::Lumber(Wood wood, SawingOption sawing\_option, Date delivery\_date,

int marking, int amount, int cost)

: Wood(wood), sawing\_option\_(sawing\_option),

delivery\_date\_(delivery\_date), marking\_(marking), amount\_(amount), cost\_(cost)

{}

// Copy constructor

Lumber::Lumber(const Lumber &copy)

: Wood(copy.GetWood()), sawing\_option\_(copy.sawing\_option\_),

delivery\_date\_(copy.delivery\_date\_), marking\_(copy.marking\_), amount\_(copy.amount\_), cost\_(copy.cost\_)

{}

Lumber::~Lumber()

{}

// Setters

Lumber &Lumber::SetWood(const Wood &wood)

{

SetTreeSpecies(wood.GetTreeSpecies());

SetWoodMoisture(wood.GetWoodMoisture());

SetWoodDensity(wood.GetWoodDensity());

return \*this;

}

Lumber &Lumber::SetSawingOption(const SawingOption sawing\_option)

{

sawing\_option\_ = sawing\_option;

return \*this;

}

Lumber &Lumber::SetDeliveryDate(const Date &delivery\_date)

{

delivery\_date\_ = delivery\_date;

return \*this;

}

Lumber &Lumber::SetMarking(const int marking)

{

marking\_ = marking;

return \*this;

}

Lumber &Lumber::SetAmount(const int amount)

{

amount\_ = amount;

return \*this;

}

Lumber &Lumber::SetCost(const int cost)

{

cost\_ = cost;

return \*this;

}

// Getters

Wood Lumber::GetWood() const { return Wood(GetTreeSpecies(), GetWoodMoisture(), GetWoodDensity()); }

SawingOption Lumber::GetSawingOption() const { return sawing\_option\_; }

Date Lumber::GetDeliveryDate() const { return delivery\_date\_; }

int Lumber::GetMarking() const { return marking\_; }

int Lumber::GetAmount() const { return amount\_; }

int Lumber::GetCost() const { return cost\_; }

// Function for printing class info

void Lumber::Print()

{

std::cout << GetWood() << ", " << SawingOptionToString(sawing\_option\_)

<< "(" << delivery\_date\_ << "), "

<< "marking: " << marking\_ << ", amount: "

<< amount\_ << ", cost: " << cost\_ << std::endl;

}

// Operators

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

{

if(this->GetWood() == obj.GetWood())

{

return Lumber(GetWood(),

sawing\_option\_,

delivery\_date\_ + obj.delivery\_date\_,

marking\_ + obj.marking\_,

amount\_ + obj.amount\_,

cost\_ + obj.cost\_);

}

return \*this;

}

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

{

return cost\_ < obj.cost\_;

}

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

{

return cost\_ > obj.cost\_;

}

std::ostream &operator<<(std::ostream &out, const Lumber &copy)

{

out << copy.GetWood() << ", " << SawingOptionToString(copy.GetSawingOption())

<< "(" << copy.GetDeliveryDate() << "), "

<< "marking: " << copy.GetMarking() << ", amount: "

<< copy.GetAmount() << ", cost: " << copy.GetCost();

return out;

}

std::istream &operator>>(std::istream &in, Lumber &copy)

{

// Set wood\_

Wood buff\_wood;

in >> buff\_wood;

copy.SetWood(buff\_wood);

// Set sawing\_option\_

int buff\_int;

std::cout << "1 - Beam" << std::endl;

std::cout << "2 - Unedged Board" << std::endl;

std::cout << "3 - Edged Board" << std::endl;

std::cout << "4 - Rail" << std::endl;

std::cout << "Enter Sawing Option(int): ";

in >> buff\_int;

copy.SetSawingOption(IntToSawingOption(buff\_int));

// Set delivery\_date\_

Date buff\_date;

in >> buff\_date;

copy.SetDeliveryDate(buff\_date);

// Set marking\_

std::cout << "Enter marking: ";

in >> buff\_int;

copy.SetMarking(buff\_int);

// Set amount\_

std::cout << "Enter amount: ";

in >> buff\_int;

copy.SetAmount(buff\_int);

// Set cost\_

std::cout << "Enter cost: ";

in >> buff\_int;

copy.SetCost(buff\_int);

return in;

}

**Клас “Workshop”**

**Header:**

#ifndef \_KP3\_INCLUDE\_KP3\_WORKSHOP\_H\_

#define \_KP3\_INCLUDE\_KP3\_WORKSHOP\_H\_

#include <KP3/lumber.h>

class Workshop

{

public:

// Default constructor(with no parameters)

Workshop();

// Constructor with parameters

Workshop(int workshop\_serial\_number);

// Copy constructor

Workshop(const Workshop &copy);

// Destructor

~Workshop();

// operator=

Workshop &operator=(const Workshop &copy);

// Setters

Workshop &SetWorkshopSerialNumber(const int workshop\_serial\_number);

Workshop &SetNewPointer(const Lumber \*pointer, const size\_t size);

// Getters

int GetShopsCount() const;

int GetWorkshopSerialNumber() const;

int GetSize() const;

// Function for printing class info

void Print();

void PrintShortly();

// Other functions

void AddLumber(const Lumber value);

bool RemoveWood(const Wood value); // return if there is wood in lumber

Lumber &operator[](const int index);

private:

static int workshops\_count\_;

int workshop\_serial\_number\_;

Lumber \*pointer\_;

size\_t size\_;

}; // Class (Workshop)

#endif // Header Guard

**Source file:**

#include <KP3/workshop.h>

#include <iostream>

int Workshop::workshops\_count\_ = 0;

// Default constructor(with no parameters)

Workshop::Workshop()

: workshop\_serial\_number\_(workshops\_count\_ + 1), size\_(0), pointer\_(nullptr)

{

workshops\_count\_++;

}

// Constructor with parameters

Workshop::Workshop(int workshop\_serial\_number)

: workshop\_serial\_number\_(workshop\_serial\_number), size\_(0), pointer\_(nullptr)

{

workshops\_count\_++;

}

// Copy constructor

Workshop::Workshop(const Workshop &copy)

: size\_(copy.size\_), workshop\_serial\_number\_(copy.workshop\_serial\_number\_)

{

pointer\_ = new Lumber[size\_];

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

{

pointer\_[i] = copy.pointer\_[i];

}

workshops\_count\_++;

}

Workshop::~Workshop()

{

if (size\_ != 0)

{

delete[] pointer\_;

}

}

// operator=

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

{

if (this != &copy) // Protection from leak of memory

{

if (size\_ != 0)

{

delete[] pointer\_;

}

pointer\_ = new Lumber[copy.size\_];

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

{

pointer\_[i] = copy.pointer\_[i];

} // For loop(i < copy.size\_)

size\_ = copy.size\_;

workshop\_serial\_number\_ = copy.workshop\_serial\_number\_;

} // Condition (this != &copy)

return \*this;

}

// Setters

Workshop &Workshop::SetWorkshopSerialNumber(const int workshop\_serial\_number)

{

workshop\_serial\_number\_ = workshop\_serial\_number;

return \*this;

}

Workshop &Workshop::SetNewPointer(const Lumber \*pointer, const size\_t size)

{

if (size\_ > 0)

{

delete[] pointer\_;

}

pointer\_ = new Lumber[size];

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

{

pointer\_[i] = pointer[i];

}

size\_ = size;

return \*this;

}

// Getters

int Workshop::GetShopsCount() const { return workshops\_count\_; }

int Workshop::GetWorkshopSerialNumber() const { return workshop\_serial\_number\_; }

int Workshop::GetSize() const { return size\_; }

// Function for printing class info

void Workshop::Print()

{

std::cout << "Workshop #" << workshop\_serial\_number\_ << std::endl;

if (size\_ > 0)

{

std::cout << "Lumber:" << std::endl;

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

{

std::cout << i + 1 << ". ";

pointer\_[i].Print();

}

}

}

void Workshop::PrintShortly()

{

std::cout << "Serial number: " << workshop\_serial\_number\_;

int sum = 0;

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

{

sum += pointer\_[i].GetAmount();

}

std::cout << ", lumbers: " << sum << std::endl;

}

// Other functions

void Workshop::AddLumber(const Lumber value)

{

if (size\_ != 0)

{

Lumber \*new\_pointer = new Lumber[size\_ + 1];

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

{

new\_pointer[i] = pointer\_[i];

}

new\_pointer[size\_] = value;

delete[] pointer\_;

pointer\_ = new\_pointer;

}

else

{

pointer\_ = new Lumber[1];

pointer\_[0] = value;

}

++size\_;

}

// return if there is toy in shop

bool Workshop::RemoveWood(const Wood value)

{

bool is\_valid\_to\_sell = false;

for (int i = 0; i < size\_ && is\_valid\_to\_sell == false; i++)

{

if (pointer\_[i].GetWood() == value)

{

is\_valid\_to\_sell = true;

if (pointer\_[i].GetAmount() - 1 > 0)

{

pointer\_[i].SetAmount(pointer\_[i].GetAmount() - 1);

}

else

{

if (size\_ != 1)

{

Lumber \*new\_pointer = new Lumber[size\_ - 1];

for (int j = 0, k = 0; j < size\_; j++)

{

if (j != i)

{

new\_pointer[k] = pointer\_[j];

k++;

} // Condition (j != i)

} // for loop (j < size\_)

delete[] pointer\_;

pointer\_ = new\_pointer;

}

else

{

delete[] pointer\_;

}

--size\_;

} // Condition (pointer\_[i].GetCounts() - 1 > 0)

} // Condition (pointer\_[i].GetToy() == value)

} // for loop (i < size\_ && is\_valid\_to\_sell == false)

return is\_valid\_to\_sell;

}

Lumber &Workshop::operator[](const int index)

{

return pointer\_[index];

}

**Main file:**

#include <KP3/workshop.h>

#include <iomanip>

#include <iostream>

int main()

{

Lumber lumber1;

Lumber lumber2;

Lumber lumber3;

lumber3.SetWood(Wood("Birch", 0, 0));

lumber3.SetCost(1000);

std::cout << "Lumber1: " << lumber1 << std::endl;

std::cout << "Lumber2: " << lumber2 << std::endl;

std::cout << "Lumber3: " << lumber3 << std::endl << std::endl;

std::cout << "Lumber1 + Lumber2 = " << (lumber1 + lumber2) << std::endl;

std::cout << "Lumber1 + Lumber3 = " << (lumber1 + lumber3) << std::endl << std::endl;

std::cout << std::setfill('-') << std::setw(100) << '\0' << std::endl << std::endl;

std::cout << "Lumber1 > Lumber3 : " << (lumber1 > lumber3) << std::endl;

std::cout << "Lumber1 < Lumber3 : " << (lumber1 < lumber3) << std::endl << std::endl;

std::cout << std::setfill('-') << std::setw(100) << '\0' << std::endl << std::endl;

Wood wood1;

Wood wood2;

wood1.SetWoodDensity(15);

std::cout << "Wood1: " << lumber1 << std::endl;

std::cout << "Wood2: " << lumber2 << std::endl << std::endl;

std::cout << "Wood1 == Wood2 : " << (wood1 == wood2) << std::endl;

std::cout << "Wood1 != Wood2 : " << (wood1 != wood2) << std::endl << std::endl;

std::cout << std::setfill('-') << std::setw(100) << '\0' << std::endl << std::endl;

Workshop workshop;

workshop.AddLumber(Lumber());

workshop.AddLumber(Lumber());

workshop.AddLumber(Lumber());

workshop.Print();

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

workshop[1].SetCost(1000);

std::cout << "Workshop after \"workshop[1].SetCost(1000)\"" << std::endl;

workshop.Print();

std::cout << std::endl << std::setfill('-') << std::setw(100) << '\0' << std::endl << std::endl;

Wood\* wood\_ptr = &lumber1;

std::cout << "Wood class is parent for Lumber class" << std::endl;

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

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

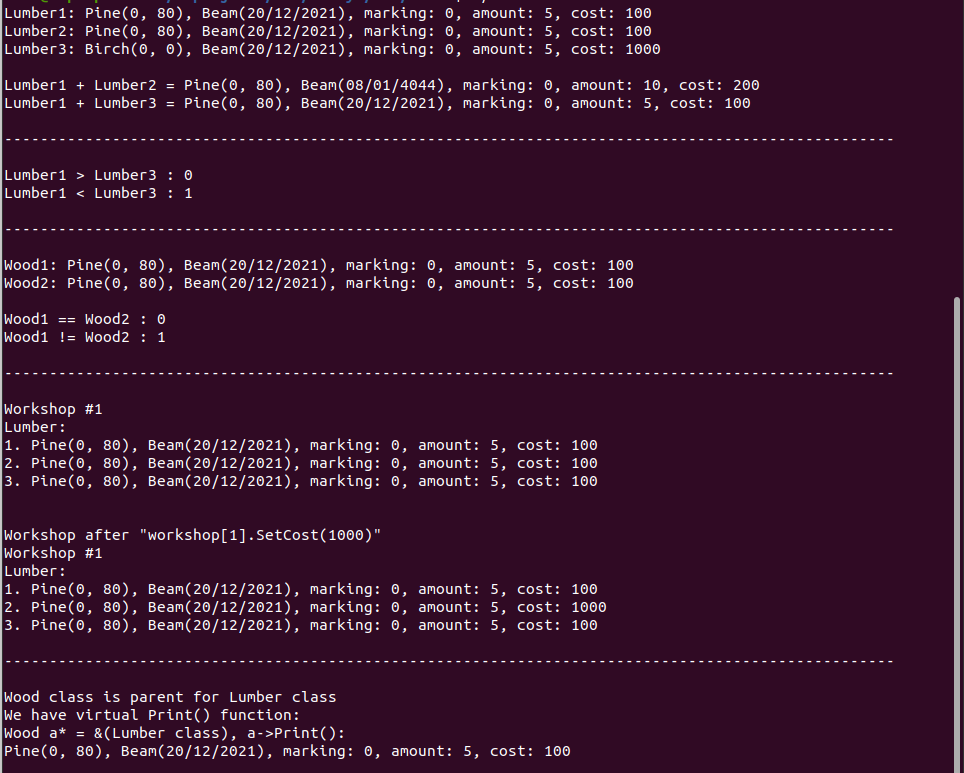
wood\_ptr->Print();

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.