МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

#### Факультет информационных технологий и компьютерной безопасности

(факультет)

### Кафедра Систем автоматизированного проектирования и информационных систем

# ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

по дисциплине Объектно-ориентированное программирование

Тема: «Виртуальные функции и полиморфизм»

Выполнили студенты группы ИСТ-222 Е.Г. Гладнева

Подпись, дата Инициалы, фамилия

Е.В. Журавлёва

Подпись, дата Инициалы, фамилия

## Принял Б.Н. Тишуков

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2023

Лабораторная работа № 7

«Виртуальные функции и полиморфизм»

1 Цель работы

Целью лабораторной работы является изучение и умение использовать виртуальные функции и полиморфизм в языке программирования C++.

1. Задание на лабораторную работу
2. Даны классы BinaryOperation (бинарный оператор) и Number (число), которые наследуются от базового класса Expression (выражение). Задача заключается в реализации базового класса Expression так, чтобы не было утечек памяти. Кроме этого необходимо продумать, какие методы стоит сделать виртуальными.
3. Необходимо добавить к иерархии из предыдущего упражнения класс-наследник FunctionCall. FunctionCall должен представлять вызов одной из двух предопределенных математических функций: sqrt — извлечение квадратного корня и abs — вычисление модуля числа. Функция идентифицируется строкой, переданной в качестве параметра в конструктор. Нужно не забыть, что у функции должен быть аргумент (которым может быть любое выражение Expression)!
4. Создать иерархию классов, используя наследование от абстрактного базового класса, согласно варианту, номер 6. В программе необходимо соблюсти принцип разделения интерфейса и реализации класса. Необходимо написать программу, иллюстрирующую применение всех методов ваших классов. Прежде чем приступить к написанию программ, нужно продумать, какие необходимы функции в каждом из классов. Также нужно продумать, что следует поместить в закрытые или защищенные переменные. Особенно необходимо обратить внимание на то, какие функции следует сделать чистыми виртуальными. Также надо предусмотреть виртуальный деструктор! В основной программе надо использовать динамическое связывание.

Все мы любим отдыхать. Но абстрактное понятие «нерабочий день» может на самом деле оказаться конкретным выходным, праздником или отпуском. Реализуйте такую иерархию. Разумеется, самый большой приоритет имеют выходные – на них могут попасть и праздники, и отпуск.

1. Ход выполнения
2. Для выполнения задания №1 для начала необходимо скопировать приведённый код и вставить его в программу, далее смотрим, какая функция повторяется в классах, находим, и понимаем, что в базовом классе «Expression» её необходимо объявить как чисто виртуальную. Проверяем дальше код, всё хорошо, остаётся в базовом классе сделать деструктор. В функциях, которые унаследованы из базового класса пишем слово «override». Проверяем правильно работает ли программа с приведёнными значениями. Всё работает правильно. Результат работы представлен на рисунке 1.

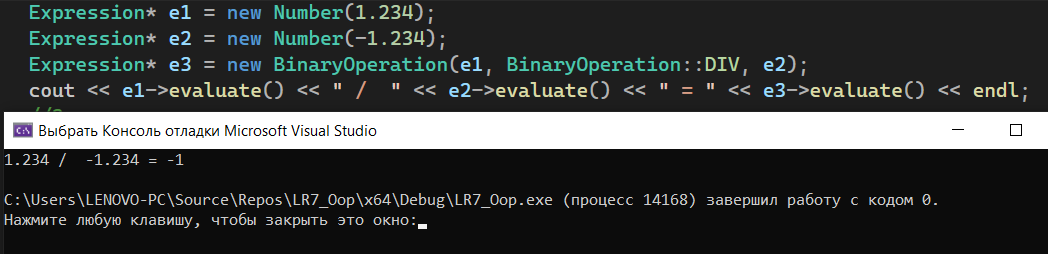


Рисунок 1 – Результат работы задания №1

1. Создаём класс – наследник FunctionCall. Для начала создаём сам класс и пишем, что он наследник. Далее создаём метод, в котором будут названия операций и что они означают. После создаём метод, который проверяет, всё ли правильно идёт. Далее создаём деструктор, после метод, в котором и происходят сами действия с числом и в конце создаём переменные типа private. Результат работы представлен на рисунке 2.

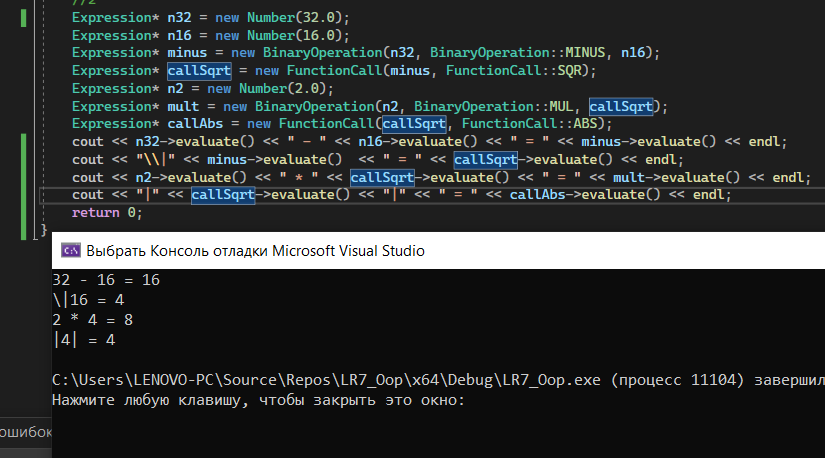


Рисунок 2 – Результат работы задания №2

1. Создаём базовый класс Non\_working\_day, в котором прописываем виртуальные методы и создаём виртуальный деструктор. Далее создаём классы наследники и в них прописываем различные методы, если они унаследованы от виртуальных, то прописываем в них слово «override». В каждом из классов прописываем их деструкторы. В основной функции вызываем их, пользуясь динамическим связыванием. Результат проделанной работы представлен на рисунке 3.

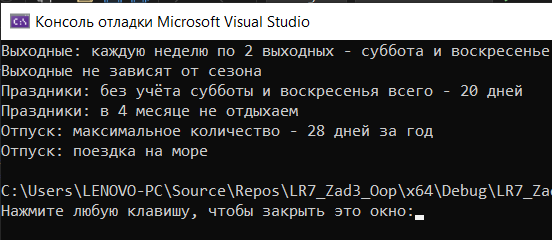


Рисунок 3 – Результат работы задания №3

1. Вывод

В ходе выполнения лабораторной работы мы научились пользоваться виртуальными функциями и полиморфизмом в языке программирования C++.

Приложение А

Листинг программного кода языка C++

#include <iostream>

#include <cassert> // assert

#include <cmath>// sqrt и fabs

using namespace std;

struct Expression

{

public:

virtual double evaluate() const = 0;

virtual ~Expression() {}

};

struct Number :public Expression

{

Number(double value) : value\_(value) {}

double value() const { return value\_; }

double evaluate() const override { return value\_; }

private:

double value\_;

};

struct BinaryOperation : Expression

{

enum {

PLUS = '+',

MINUS = '-',

DIV = '/',

MUL = '\*'

};

BinaryOperation(Expression const\* left, int op,

Expression const\* right) :left\_(left), op\_(op), right\_(right)

{

assert(left\_ && right\_);

}

~BinaryOperation()

{

delete left\_;

delete right\_;

}

Expression const\* left() const { return left\_; }

Expression const\* right() const { return right\_; }

int operation() const { return op\_; }

double evaluate() const

{

double left = left\_->evaluate();

double right = right\_->evaluate();

switch (op\_)

{

case PLUS: return left + right;

case MINUS: return left - right;

case DIV: return left / right;

case MUL: return left \* right;

}

assert(0);

return 0.0;

}

private:

Expression const\* left\_;

Expression const\* right\_;

int op\_;

};

struct FunctionCall : Expression

{

enum {

SQR = '\|',

ABS = 'mod'

};

FunctionCall(Expression const\* chislo, int op) :chislo\_(chislo), op\_(op)

{

assert(chislo\_);

}

~FunctionCall()

{

delete chislo\_;

}

Expression const\* chislo() const { return chislo\_; }

int operation() const { return op\_; }

double evaluate() const

{

double chislo = chislo\_->evaluate();

switch (op\_)

{

case SQR: return sqrt(chislo);

case ABS: return abs(chislo);

}

assert(0);

return 0.0;

}

private:

Expression const\* chislo\_;

int op\_;

};

int main()

{

//1

Expression\* e1 = new Number(1.234);

Expression\* e2 = new Number(-1.234);

Expression\* e3 = new BinaryOperation(e1, BinaryOperation::DIV, e2);

cout << e1->evaluate() << " / " << e2->evaluate() << " = " << e3->evaluate() << endl;

//2

Expression\* n32 = new Number(32.0);

Expression\* n16 = new Number(16.0);

Expression\* minus = new BinaryOperation(n32, BinaryOperation::MINUS, n16);

Expression\* callSqrt = new FunctionCall(minus, FunctionCall::SQR);

Expression\* n2 = new Number(2.0);

Expression\* mult = new BinaryOperation(n2, BinaryOperation::MUL, callSqrt);

Expression\* callAbs = new FunctionCall(callSqrt, FunctionCall::ABS);

cout << n32->evaluate() << " - " << n16->evaluate() << " = " << minus->evaluate() << endl;

cout << "\\|" << minus->evaluate() << " = " << callSqrt->evaluate() << endl;

cout << n2->evaluate() << " \* " << callSqrt->evaluate() << " = " << mult->evaluate() << endl;

cout << "|" << callSqrt->evaluate() << "|" << " = " << callAbs->evaluate() << endl;

return 0;

}

Приложение B

Листинг программного кода языка C++

#include <iostream>

using namespace std;

class Non\_working\_day

{

public:

virtual void count\_day() = 0;

virtual void season() = 0;

virtual ~Non\_working\_day() {};

};

class Weekends : public Non\_working\_day //выходные

{

public:

void count\_day() override

{

cout << "Выходные: каждую неделю по 2 выходных - суббота и воскресенье" << endl;

}

void season() override

{

cout << "Выходные не зависят от сезона" << endl;

}

~Weekends() {};

};

class Holidays : public Non\_working\_day //праздники

{

public:

void count\_day() override

{

cout << "Праздники: без учёта субботы и воскресенья всего - 20 дней" << endl;

}

void season() override

{

cout << "Самый длительный промежуток праздников - 9 дней, в январе;\nСамый короткий промежуток праздников - 2 дня, в ноябре" << endl;

}

void holidays(int may) //кол-во дней в зависимости от праздника

{

cout << "Праздники: в " << may << " месяце ";

if (may == 1)

cout << "новогодние каникулы - 9 дней\n";

if (may == 2)

cout << "23 февраля - 3 дня\n";

if (may == 3)

cout << "8 марта - 3 дня\n";

if (may == 10)

cout << "День народного единства - 2 дня\n";

if ((4 <= may && may <= 9) || (11 <= may && may <= 12))

cout << "не отдыхаем\n";

}

~Holidays() {};

};

class Vacation : public Non\_working\_day //Отпуск

{

public:

void count\_day() override

{

cout << "Отпуск: максимальное количество - 28 дней за год" << endl;

}

void season() override

{

cout << "Отпуск может быть взят в любой сезон" << endl;

}

void occasion() //повод

{

cout << "Отпуск: поездка на море" << endl;

}

~Vacation() {};

};

int main()

{

setlocale(LC\_ALL, "RUS");

Weekends weekends;

Non\_working\_day\* day = &weekends;

day->count\_day();

day->season();

Holidays holidays1;

Non\_working\_day\* day1 = &holidays1;

day1->count\_day();

int may = 4;

holidays1.holidays(may);

Vacation vacation2;

Non\_working\_day\* day2 = &vacation2;

day2->count\_day();

vacation2.occasion();

return 0;

}