Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра И5 «Информационные системы и программная инженерия»

**Лабораторная работа №2**по дисциплине «Программирование на языке высокого уровня»  
по теме «Наследование»

Выполнил:  
Студент Альков В. С.  
Группа И407Б  
  
Преподаватель:  
Кимсанбаев К. А.

Санкт-Петербург  
2021 г.

Задача: Описать три класса: базовый класс «Строка» и производные от него класс «Строка-

идентификатор» и класс, заданный индивидуальным вариантом. Обязательные для всех классов методы: конструктор без параметров, конструктор, принимающий в качестве параметра Си-строку, конструктор копирования, деструктор, перегрузка операции присваивания «=». Во всех методах всех классов предусмотреть печать сообщения, содержащего имя метода. Для конструкторов копирования каждого класса дополнительно предусмотреть диагностическую печать количества его вызовов, рекомендуется использовать статические члены класса.

Поля класса «Строка»: указатель на блок динамически выделенной памяти для размещения символов строки, длина строки в байтах. Обязательные методы, помимо вышеуказанных: конструктор, принимающий в качестве параметра символ (char), функция получения длины строки.

Строки класса «Строка-идентификатор» строятся по правилам записи идентификаторов в Си, и могут включать в себя только те символы, которые могут входить в состав Си-идентификаторов. Если исходные данные противоречат правилам записи идентификатора, то создается пустая «Строка-идентификатор».

Помимо обязательных компонентов классов, указанных в общей постановке задачи и в вариативной его части, при необходимости можно добавить дополнительные поля и методы.

Написать тестовую программу, которая должна:

* динамически выделить память под массив указателей на базовый класс (4-6 шт.);
* в режиме диалога заполнить этот массив указателями на производные классы, при этом экземпляры производных классов должны создаваться динамически с заданием начальных значений;
* для созданных экземпляров производных классов выполнить проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Режим диалога должен обеспечиваться с помощью иерархического меню. Основные пункты:

1. «Инициализация». Подпункты:

1.1. «Число элементов». Задает число элементов в массиве указателей на базовый класс. После ввода числа элементов пользоваться этим пунктом меню запрещается.

1.2. «Начальное значение». С помощью этого пункта меню можно задать номер элемента, его тип и начальное значение. Задавать начальные значения и работать с другими пунктами меню запрещается до тех пор, пока не будет задано число элементов. Допускается задать новое начальное значение несколько раз.

2. «Тестирование». Подпункты:

2.1. «Строка».

2.2. «Строка-идентификатор».

2.3. Класс, соответствующий варианту задания.

2.4. «Задать операнды».

После выбора одного из этих пунктов меню предлагается выбрать один из методов из списка всех обязательных методов (кроме конструкторов и деструкторов), связанных с выбранным подпунктом.

3. Выход.

**Текст вариативной части задания**

Дополнительные методы для класса «Строка-идентификатор»: перевод всех символов строки (кроме цифр) в нижний регистр, переопределение операции вычитания «-» (из первого операнда удаляются все символы, входящие во второй операнд).

Производный от «Строки» класс «Десятичная строка».

Строки данного класса могут содержать только символы десятичных цифр и символы «-» и «+», задающие знак числа, которые могут находиться только в первой позиции числа, при отсутствии знака число считается положительным. Если в составе инициализирующей

строки будут встречены любые символы, отличные от допустимых, «Десятичная строка» принимает нулевое значение. Содержимое данных строк рассматривается как десятичное число.

Обязательные методы: определение, можно ли представить данное число в формате long, перегрузка операций вычитания «-» и умножения «\*» для получения разности и произведения двух десятичных чисел.

**Класс «Строка»**

**string.h**

#ifndef STRING\_H

#define STRING\_H

class Decimal;

class Identifier;

class String

{

/\*поля кол-во копирования, массив символов для строки, размер строки\*/

static int copyCount;

protected:

char \*str;

int size;

public:

/\*конструкторы\*/

String();

String(char c);

String(char \*strSource);

String(const char \*strSource) : String((char\*) strSource){};

/\*конструктор копирования\*/

String(const String& object);

/\*декструктор\*/

virtual ~String();

/\*операторы присваивания\*/

String operator=(const String& object);

String operator=(char \*strSource);

String operator=(const char \*strSource);

/\*метод печати кол-ва копирований\*/

virtual void printCopyCount();

/\*метод печати строки\*/

virtual void print();

/\*метод получения длины строки\*/

int strLenght();

/\*метод определения какого типа строка\*/

virtual int getType();

friend Decimal;

friend Identifier;

};

#endif // STRING\_H

**string.cpp**

#include "string.h"

#include <iostream>

#include <cstring>

#define PrintMethodName1 1

String::String()

{

if(PrintMethodName1)

std::cout << "String::String()\n";

/\*создаем пустую строку\*/

str = new char[1];

\*str = '\0';

size = 1;

}

String::String(char c)

{

if(PrintMethodName1)

std::cout << "String::String(char c)\n";

str = new char[2];

\*str = c;

\*(str+1) = '\0';

size = 2;

}

String::String(char \*strSource)

{

if(PrintMethodName1)

std::cout << "String::String(char \*strSource)\n";

size = strlen(strSource)+1;

str = new char[size];

strcpy(str, strSource);

}

String::String(const String& object)

{

if(PrintMethodName1)

std::cout << "String::String(const String& object)\n";

size = object.size;

str = new char[size];

strcpy(str, object.str);

copyCount++;

}

String::~String()

{

if(PrintMethodName1)

std::cout << "String::~String()\n";

delete[] str;

str = NULL;

}

String String::operator=(const String& object)

{

if(PrintMethodName1)

std::cout << "String::operator=(const String& object)\n";

delete[] str;

size = object.size;

str = new char[size];

strcpy(str, object.str);

return \*this;

}

String String::operator=(char \*strSource)

{

if(PrintMethodName1)

std::cout << "String::operator=(char \*strSource)\n";

/\*освобождаем память\*/

delete[] str;

/\*если переданная строка не нулевая\*/

if(strSource)

{

/\*то копируем ее\*/

size = strlen(strSource);

str = new char[size];

strcpy(str, strSource);

}

else

{

/\*иначе создаем пустую\*/

str = new char[1];

\*str = '\0';

size = 1;

};

return \*this;

}

String String::operator=(const char \*strSource)

{

if(PrintMethodName1)

std::cout << "String::operator=(const char \*strSource)\n";

delete[] str;

size = strlen(strSource)+1;

str = new char[size];

strcpy(str, strSource);

return \*this;

}

void String::print()

{

if(PrintMethodName1)

std::cout << "String::print()\n";

if(str)

std::cout << str <<"\n";

}

int String::strLenght()

{

if(PrintMethodName1)

std::cout << "String::strLenght()\n";

if(str)

size = strlen(str)+1;

return size-1;

}

int String::getType()

{

if(PrintMethodName1)

std::cout << "String::getType()\n";

return 1;

}

int String::copyCount=0;

void String::printCopyCount()

{

if(PrintMethodName1)

std::cout << "String::printCopyCount()\n";

std::cout<<copyCount;

};

**identifier.h**

#ifndef IDENTIFIER\_H

#define IDENTIFIER\_H

#include "string.h"

/\*ф-ия проверки возможности создания из строки строки-идентификатора\*/

int checkPossibility(char \*tmp);

class Identifier: public String

{

static int copyCount;

friend int checkPossibility(char \*tmp);

public:

/\*конструкторы\*/

Identifier(char \*strSource);

Identifier(const char \*strSource) : Identifier((char\*) strSource){};

/\*конструктор копирования\*/

Identifier(const Identifier& object);

/\*конструкторы копирования для приведения типа\*/

Identifier(const String& object);

Identifier(const Decimal& object);

/\*метод приведения к нижнему регистру\*/

Identifier toLowerCase();

/\*деструктор\*/

~Identifier();

/\*оператор вычитания символов\*/

Identifier operator-(const Identifier& object);

/\*метод определения типа строки\*/

int getType();

/метод печати кол-ва копирований/

void printCopyCount();

};

#endif // IDENTIFIER\_H

**identifier.cpp**

#include "identifier.h"

#include <cstring>

#include <iostream>

#define PrintMethodName2 1

int checkPossibility(char \*tmp);

int Identifier::copyCount = 0;

Identifier::Identifier(const Identifier& object) : String(object)

{

copyCount++;

}

Identifier::Identifier(const String& object) : Identifier(object.str)

{

copyCount++;

}

Identifier::Identifier(const Decimal& object) : String()

{

copyCount++;

}

int Identifier::getType()

{

if(PrintMethodName2)

std::cout << "Identifier::getType()\n";

return 2;

}

void Identifier::printCopyCount()

{

if(PrintMethodName2)

std::cout << "Identifier::printCopyCount()\n";

std::cout<<copyCount;

}

int checkPossibility(char \*tmp)

{

if(PrintMethodName2)

std::cout << "friend Identifier int checkPossibility()\n";

/\*проверка совпадения строки с ключевыми словами, если истина, то строка не подходит\*/

if(strcmp(tmp, "alignas") == 0 || strcmp(tmp, "alignof") == 0 || strcmp(tmp, "and") == 0 || strcmp(tmp, "and\_eq") == 0 || strcmp(tmp, "asm") == 0 || strcmp(tmp, "auto") == 0 || strcmp(tmp, "bitand") == 0 || strcmp(tmp, "bitor") == 0 || strcmp(tmp, "bool") == 0 || strcmp(tmp, "break") == 0 || strcmp(tmp, "case") == 0 || strcmp(tmp, "catch") == 0 || strcmp(tmp, "char") == 0 || strcmp(tmp, "char16\_t") == 0 || strcmp(tmp, "char32\_t") == 0 || strcmp(tmp, "class") == 0 || strcmp(tmp, "compl") == 0 || strcmp(tmp, "const") == 0 || strcmp(tmp, "constexpr") == 0 || strcmp(tmp, "const\_cast") == 0 || strcmp(tmp, "continue") == 0 || strcmp(tmp, "decltype ") == 0 || strcmp(tmp, "default") == 0 || strcmp(tmp, "delete") == 0 || strcmp(tmp, "do") == 0 || strcmp(tmp, "double") == 0 || strcmp(tmp, "dynamic\_cast") == 0 || strcmp(tmp, "else") == 0 || strcmp(tmp, "enum") == 0 || strcmp(tmp, "explicit") == 0 || strcmp(tmp, "export") == 0 || strcmp(tmp, "extern") == 0 || strcmp(tmp, "false") == 0 || strcmp(tmp, "float") == 0 || strcmp(tmp, "for") == 0 || strcmp(tmp, "friend") == 0 || strcmp(tmp, "goto") == 0 || strcmp(tmp, "if") == 0 || strcmp(tmp, "inline") == 0 || strcmp(tmp, "int") == 0 || strcmp(tmp, "long") == 0 || strcmp(tmp, "mutable") == 0 || strcmp(tmp, "namespace ") == 0 || strcmp(tmp, "new") == 0 || strcmp(tmp, "noexcept") == 0 || strcmp(tmp, "not") == 0 || strcmp(tmp, "not\_eq") == 0 || strcmp(tmp, "nullptr") == 0 || strcmp(tmp, "operator") == 0 || strcmp(tmp, "or") == 0 || strcmp(tmp, "or\_eq") == 0 || strcmp(tmp, "private") == 0 || strcmp(tmp, "protected") == 0 || strcmp(tmp, "public") == 0 || strcmp(tmp, "register") == 0 || strcmp(tmp, "reinterpret\_cast") == 0 || strcmp(tmp, "return") == 0 || strcmp(tmp, "short") == 0 || strcmp(tmp, "signed") == 0 || strcmp(tmp, "sizeof") == 0 || strcmp(tmp, "static") == 0 || strcmp(tmp, "static\_assert") == 0 || strcmp(tmp, "static\_cast") == 0 || strcmp(tmp, "struct") == 0 || strcmp(tmp, "switch") == 0 || strcmp(tmp, "template") == 0 || strcmp(tmp, "this") == 0 || strcmp(tmp, "thread\_local") == 0 || strcmp(tmp, "throw") == 0 || strcmp(tmp, "true") == 0 || strcmp(tmp, "try") == 0 || strcmp(tmp, "typedef") == 0 || strcmp(tmp, "typeid") == 0 || strcmp(tmp, "typename") == 0 || strcmp(tmp, "union") == 0 || strcmp(tmp, "unsigned") == 0 || strcmp(tmp, "using") == 0 || strcmp(tmp, "virtual") == 0 || strcmp(tmp, "void") == 0 || strcmp(tmp, "volatile") == 0 || strcmp(tmp, "wchar\_t") == 0 || strcmp(tmp, "while") == 0 || strcmp(tmp, "xor") == 0 || strcmp(tmp, "xor\_eq") == 0)

return 0;

/\*если первый символ не латинская буква или нижнее подчеркивание, то строка не подходит по правилам\*/

if (\*tmp != '\_' && (\*tmp<'A' || \*tmp>'Z') && (\*tmp<'a' || \*tmp>'z'))

return 0;

/\*проверка в цикле, если символ не является латинской буквой или нижним подчеркиванием или цифрой, то строка не подходит\*/

while(\*++tmp)

if(\*tmp != '\_' && (\*tmp<'A' || \*tmp>'Z') && (\*tmp<'a' || \*tmp>'z') && (\*tmp<'1' || \*tmp>'9'))

return 0;

return 1;

}

Identifier::~Identifier()

{

if(PrintMethodName2)

std::cout << "Identifier::~Identifier()\n";

delete[] str;

str = NULL;

}

Identifier::Identifier(char \*strSource)

{

if(PrintMethodName2)

std::cout << "Identifier::Identifier(char \*strSource)\n";

/\*если строка подходит по правилам идентификатора, то копируем ее, если нет, то пустая строка уже создана при создании объекта\*/

if(checkPossibility(strSource))

{

delete[] str;

size = strlen(strSource)+1;

str = new char[size];

strcpy(str, strSource);

};

}

Identifier Identifier::toLowerCase()

{

if(PrintMethodName2)

std::cout << "Identifier::toLowerCase()\n";

/\*в цикле, пока не дошли до конца строки, если не цифра и не нижнее подчёркивание, то понижаем регистр\*/

for(int i=0; str[i]!='\0'; i++)

if((str[i]<'0' || str[i]>'9') && str[i]!='\_')

str[i] = std::tolower(str[i]);

return \*this;

}

Identifier Identifier::operator-(const Identifier& object)

{

if(PrintMethodName2)

std::cout << "Identifier::operator-(const Identifier& object)\n";

char \*p, \*tmp = new char[size];

strcpy(tmp, str);

for(int i=0; object.str[i]; i++)

if((p = strchr(tmp, object.str[i])) != NULL)

for(; \*p; \*p=\*(p+1), p++);

return Identifier(tmp);

}

**decimal.h**

#ifndef DECIMAL\_H

#define DECIMAL\_H

#include "string.h"

/\*ф-ия проверки строки на число\*/

int checkPossibilityDecimal(char \*str);

class Decimal : public String

{

/поля кол-ва копирования, знак числа/

static int copyCount;

int sign;

public:

/\*конструкторы\*/

Decimal();

Decimal(char\* strSource);

Decimal(const char\* strSource) : Decimal((char\*)strSource){};

/\*конструктор копирования\*/

Decimal(const Decimal& object);

/\*конструктор копирования для приведения типа\*/

Decimal(const String& object);

Decimal(const Identifier& object);

/\*деструктор\*/

~Decimal();

/\*метод проверки представимости в long\*/

int isLong();

/\*оператор вычитания\*/

Decimal operator-(const Decimal& object);

/\*оператор умножения\*/

Decimal operator\*(const Decimal& object);

/\*оператор сложения\*/

Decimal operator+(const Decimal& object);

/\*операторы присваивания\*/

Decimal operator=(char\* strSource);

Decimal operator=(const char \*strSource);

Decimal operator=(const Decimal& object);

/\*метод смены знака\*/

Decimal makeNegative();

/\*метод печати\*/

void print();

friend void checkPossibilityDecimal(Decimal\* obj, char \*strSource);

/\*метод определения типа строки\*/

int getType();

/\*метод печати кол-ва копирования\*/

void printCopyCount();

};

#endif //DECIMAL\_H

**decimal.cpp**

#include "decimal.h"

#include <cstring>

#include <iostream>

#define PrintMethodName3 1

int Decimal::copyCount = 0;

int Decimal::getType()

{

if(PrintMethodName3)

std::cout << "Decimal::getType()\n";

return 3;

}

Decimal::Decimal(const String& object): Decimal(object.str)

{

copyCount++;

}

Decimal::Decimal(const Identifier& object): Decimal()

{

copyCount++;

}

void Decimal::printCopyCount()

{

if(PrintMethodName3)

std::cout << "Decimal::printCopyCount()\n";

std::cout<<copyCount;

}

void checkPossibilityDecimal(Decimal\* obj, char \*strSource)

{

if(PrintMethodName3)

std::cout << "friend checkPossibilityDecimal(Decimal\* obj, char \*strSource)\n";

int i;

/\*очищаем строку, присваиваем 0, знак +\*/

delete[] obj->str;

obj->size = 2;

obj->str = new char[obj->size];

\*obj->str = '0';

\*(obj->str+1) = '\0';

obj->sign = 1;

/\*если первый символ не + или – и не цифра, то не строка не подходит\*/

if(\*strSource!='+' && \*strSource!='-' && (\*strSource<'0' || \*strSource>'9'))

return;

/\*если первый символ “-”, то ставим объекту знак “-”\*/

if (\*strSource == '-')

{

obj->sign = -1;

strSource++;

}

else if (\*strSource == '+')

strSource++;

/\*убираем лишние нули, если они есть\*/

while(\*strSource=='0'&&\*(strSource + 1))

strSource++;

/\*проходим по строке, если встречен знак или не цифра, то строка не подходит\*/

for(i=0; strSource[i]; i++)

if(strSource[i]=='+' || strSource[i]=='-' || strSource[i]<'0' || strSource[i]>'9')

return;

/\*очищаем строку из 0, и копируем подходящую\*/

delete[] obj->str;

obj->size = strlen(strSource)+1;

obj->str = new char[obj->size];

strcpy(obj->str, strSource);

}

Decimal::Decimal()

{

if(PrintMethodName3)

std::cout << "Decimal::Decimal()\n";

delete[] str;

size = 2;

str = new char[size];

\*str = '0';

\*(str+1) = 0;

sign = 1;

}

Decimal::Decimal(char\* strSource)

{

if(PrintMethodName3)

std::cout << "Decimal::Decimal(char\* strSource)\n";

checkPossibilityDecimal(this, strSource);

}

Decimal::Decimal(const Decimal& object) : String(object)

{

if(PrintMethodName3)

std::cout << "Decimal::Decimal(const Decimal& object)\n";

sign = object.sign;

copyCount++;

}

Decimal::~Decimal()

{

if(PrintMethodName3)

std::cout << "Decimal::~Decimal()\n";

delete[] str;

str = NULL;

}

Decimal Decimal::operator=(const Decimal& object)

{

if(PrintMethodName3)

std::cout << "Decimal::operator=(const Decimal& object)\n";

delete[] str;

size = object.size;

str = new char[size];

strcpy(str, object.str);

sign = object.sign;

return \*this;

}

Decimal Decimal::operator=(char \*strSource)

{

if(PrintMethodName3)

std::cout << "Decimal::operator=(char \*strSource)\n";

checkPossibilityDecimal(this, strSource);

return \*this;

}

Decimal Decimal::operator=(const char \*strSource)

{

if(PrintMethodName3)

std::cout << "Decimal::operator=(char \*strSource)\n";

checkPossibilityDecimal(this, (char\*)strSource);

return \*this;

}

Decimal Decimal::operator+(const Decimal& object)

{

if(PrintMethodName3)

std::cout << "Decimal::operator+(const Decimal& object))\n";

int i, j;

Decimal \*obj1 = this;

const Decimal \*obj2 = &object;

/\*ставим большее число по модулю первым\*/

if((obj1->size == obj2->size && strcmp(obj1->str, obj2->str)<0) || obj1->size < obj2->size)

{

obj1 = (Decimal\*)&object;

obj2 = this;

};

/\*если знаки не равны, то меняем знак у отрицательного и вычитаем его из положительного\*/

if(obj1->sign == 1 && obj2->sign == -1) return (Decimal)\*obj1 - Decimal(\*obj2).makeNegative();

if(obj1->sign == -1 && obj2->sign == 1) return (Decimal)\*obj2 - Decimal(\*obj1).makeNegative();

/\*если знаки равны, то производим сумму в столбик с помощью символьной арифметики\*/

char \*result = new char[obj1->size+1];

strcpy(result, obj1->str);

for(i = obj1->size; i>0; result[i] = result[i-1], i--);

result[0] = '0';

for(i=obj1->size-1, j=obj2->size-2; j>=0; j--, i--)

if ((result[i] = (result[i]-'0') + (obj2->str[j] - '0') + '0') > '9')

{

/\*если разряд переполнен\*/

result[i]-=10;

result[i-1]++;

};

for(; i>=0; i--)

if (result[i] > '9')

{

result[i]-=10;

result[i-1]++;

}

else

break;

Decimal res (result);

res.sign = obj1->sign;

delete[] result;

return res;

}

Decimal Decimal::operator-(const Decimal& object)

{

if(PrintMethodName3)

std::cout << "Decimal::operator-(const Decimal& object)\n";

int i, j,flag=1;

Decimal \*obj1 = this;

const Decimal \*obj2 = &object;

/\*если числа равны, то разность будет равна 0\*/

if(obj1->size == obj2->size && strcmp(obj1->str, obj2->str)==0 && obj1->sign == obj2->sign)

return Decimal();

/\*ставим большее число по модулю первым\*/

if((obj1->size == obj2->size && strcmp(obj1->str, obj2->str)<0) || obj1->size < obj2->size)

{

obj1 = (Decimal\*)&object;

obj2 = this;

/\*флаг, что перестановка произведена\*/

flag = -1;

};

/\*если знаки не равны\*/

if(obj1->sign != obj2->sign)

{

/\*если была перестановка, то надо поменять знак у первого числа, если не было, то у второго \*/

if(flag==-1)

return (Decimal)Decimal(\*obj1).makeNegative() + \*obj2;

else

return \*obj1 + Decimal(\*obj2).makeNegative();

};

char\* result = new char[obj1->size];

strcpy(result, obj1->str);

/\*производим разность в столбик с помощью символьной арифметики\*/

for(i=obj1->size-2, j=obj2->size-2; j>=0; j--, i--)

if ((result[i] = (result[i]-'0') - (obj2->str[j] - '0') + '0') < '0')

{

result[i]+=10;

result[i-1]--;

};

for(; i>=0; i--)

if (result[i] < '0')

{

/\*если не хватает, то берем у старшего разряда\*/

result[i]+=10;

result[i-1]--;

}

else

break;

Decimal res (result);

res.sign = obj1->sign\*flag;

delete[] result;

return res;

}

void Decimal::print()

{

if(PrintMethodName3)

std::cout << "Decimal::print()\n";

char c;

if (sign == 1) c='+';

if (sign == -1) c='-';

if(c=='-')

std::cout << c << str <<"\n";

else

std::cout << str <<"\n";

}

Decimal Decimal::makeNegative()

{

if(PrintMethodName3)

std::cout << "Decimal::makeNegative()\n";

sign\*=-1;

return \*this;

}

Decimal Decimal::operator\*(const Decimal& object)

{

if(PrintMethodName3)

std::cout << "Decimal::operator\*(const Decimal& object)\n";

int i, j, k, tmp, c=0, maxsize = size + object.size - 1, offset = maxsize - 2;

unsigned char \*str3 = new unsigned char[maxsize];

str3[maxsize-1] = 0;

Decimal res;

/\*умножение в столбик, str3 – результат умножение числа в объекте на одну цифру, переданного объекта\*/

for(i = object.size-2; i>=0; c++, offset = maxsize-2-c, i--)

{

/\*обнуляем str3\*/

for(k=0; k<maxsize-1; k++)

str3[k] = '0';

/\*умножаем число на цифру переданного объекта\*/

for(j=size-2; j>=0; j--, offset--)

{

if ((str3[offset] = str3[offset] + (str[j]-'0') \* (object.str[i]-'0')) > '9')

{

tmp = str3[offset] - '0';

str3[offset] = tmp%10 + '0';

str3[offset-1] += tmp/10;

};

};

for(;offset>0; offset--)

if (str3[offset] > '9')

{

tmp = str3[offset]-'0';

str3[offset] = tmp%10+'0';

str3[offset-1]+= tmp/10;

}

else

break;

/\*прибавляем результат умножения\*/

res = res + Decimal((char\*)str3);

};

delete[] str3;

/\*знак итогового числа равен перемножению знаков операндов\*/

res.sign = sign\*object.sign;

return res;

}

int Decimal::isLong()

{

if(PrintMethodName3)

std::cout << "Decimal::isLong()\n";

char longMax[30], longMin[30];

sprintf(longMax, "%ld%c", LONG\_MAX, 0);

sprintf(longMin, "%ld%c", LONG\_MIN, 0);

/\*находим представимость с помощью вычитания\*/

if(sign<0)

if((\*this - Decimal(longMin)).sign>0)

return 1;

if(sign>0)

if((Decimal(longMax) - \*this).sign>0)

return 1;

return 0;

}

**Тестирующая программа main.cpp**

#include "string.h"

#include "identifier.h"

#include "decimal.h"

#include <iostream>

#include <locale.h>

using namespace std;

int main()

{

/\*menu – главное, menu1 – подменю; numbers – массив индексов объектов разных типов; countString – кол-во строк; countIdentifier – кол-во идентификаторов; countDecimal – кол-во десят. чисел; operand1, operand2 – индексы операндов; count – кол-во объектов; typeOperand1, typeOperand2 – типы операндов\*/

int menu, menu1, flag = 0, \*\*numbers, i=0,countString=0,countIdentifier=0,countDecimal=0, operand1, operand2, count = 0, typeOperand1, typeOperand2;

String \*\*p = 0;

char start[80] ="", operator1;

setlocale(LC\_ALL, "rus");

system("chcp 1251");

do

{

system("cls");

cout<<"1. Инициализация\n";

cout<<"2. Тестрование\n";

cout<<"3. Выход\n";

cin >> menu1;

getchar();

system("cls");

switch(menu1)

{

case 1: cout<<"1. Число элементов\n";

cout<<"2. Начальное значение\n";

cin >> menu;

getchar();

switch(menu)

{

case 1: if(flag)

{

cout<<"Задать число элементов можно только один раз\n";

getchar();

break;

}

cout<<"Введите число элементов: ";

cin >> count;

getchar();

if(count > 0)

{

/\*выделяем память под указатели на объекты и массив индексов, ставим флаг = 1, что память выделена\*/

p = new String\*[count];

numbers = new int\*[3];

numbers[0] = new int[count];

numbers[1] = new int[count];

numbers[2] = new int[count];

flag = 1;

}

else

cout<<"Неправильный ввод\n";

break;

case 2: if(!flag)

{

cout << "Задайте кол-во элементов\n";

getchar();

break;

};

cout<<"1. Для всех элементов\n";

cout<<"2. Изменить значение элемента\n";

cin >> menu;

getchar();

switch(menu)

{

case 1: if(flag == 2)

{

cout<<"Задать начальное значение можно только один раз\n";

getchar();

break;

};

for(i=0; i<count; i++)

{

system("cls");

cout<<"Элемент " << i+1<<"\n";

cout<<"Выберите тип\n";

cout<<"1. Строка\n";

cout<<"2. Строка-идентификатор\n";

cout<<"3. Десятичное число\n";

cin>> menu;

getchar();

cout<<"Начальное значение элемента: ";

cin.getline(start, 79);

/\*создаем объекты и запоминаем индексы\*/

switch(menu)

{

case 1: p[i] = new String(start);

numbers[0][countString++] = i;

break;

case 2: p[i] = new Identifier(start);

numbers[1][countIdentifier++] = i;

break;

case 3: p[i] = new Decimal(start);

numbers[2][countDecimal++] = i;

break;

default: cout<< "Неправильный ввод." <<

"Элемент будет типа Строка";

p[i] = new String(start);

numbers[0][countString++] = i;

getchar();

break;

};

getchar();

};

flag=2;

break;

case 2: if(flag!=2)

{

cout<< "Элементам не было присвоено начальное"<<

" значение в пунке 1, нельзя изменить значение\n";

getchar();

break;

};

cout<<"Введите номер элемента: ";

cin>> menu;

getchar();

if(menu>0 && menu<=count)

{

cout<<"Новое значение элемента: ";

cin.getline(start, 79);

int type = p[--menu]->getType();

if(type == 2)

\*(static\_cast<Identifier\*>(p[menu])) = start;

else

if(type == 3)

\*(static\_cast<Decimal\*>(p[menu])) = start;

else

\*p[menu] = start;

}

else

cout<<"Неправильный ввод\n";

getchar();

break;

default: cout<<"Неправильный ввод"; break;

};

break;

default: cout<< "Неправильный ввод"; break;

};

break;

case 2: if(flag !=2)

{

cout<<"Не пройден пункт 1 (Инициализация)\n";

getchar();

break;

};

cout<<"1. Строка\n";

cout<<"2. Строка-идентификатор\n";

cout<<"3. Десятичное число\n";

cout<<"4. Задать операнды\n";

cin>> menu;

getchar();

switch(menu)

{

case 1: cout<<"1. Печать\n";

cout<<"2. Длина строки\n";

cout<<"3. Тип\n";

cout<<"4. Кол-во копирования\n";

cin>> menu;

getchar();

if(menu>0 && menu<5)

for(i=0; i<countString; i++)

{

switch(menu)

{

case 1: p[numbers[0][i]]->print(); break;

case 2: cout<<p[numbers[0][i]]->strLenght()<<"\n"; break;

case 3: cout<<p[numbers[0][i]]->getType()<<"\n"; break;

case 4: p[numbers[0][i]]->printCopyCount(); break;

};

}

else

cout<< "Неправильный ввод\n";

break;

case 2: cout<<"1. Печать\n";

cout<<"2. Длина строки\n";

cout<<"3. Тип\n";

cout<<"4. Кол-во копирования\n";

cout<<"5. К нижнему регистру\n";

cin>> menu;

getchar();

if(menu>0 && menu<6)

for(i=0; i<countIdentifier; i++)

{

cout<<"Элемент "<<numbers[1][i]+1<<": ";

switch(menu)

{

case 1: p[numbers[1][i]]->print(); break;

case 2: cout<<p[numbers[1][i]]->strLenght()<<"\n"; break;

case 3: cout<<p[numbers[1][i]]->getType()<<"\n"; break;

case 4: p[numbers[1][i]]->printCopyCount(); break;

case 5:

(static\_cast<Identifier\*>(p[numbers[1][i]])->toLowerCase()).print();

break;

};

}

else

cout<< "Неправильный ввод\n";

break;

case 3: cout<<"1. Печать\n";

cout<<"2. Длина строки\n";

cout<<"3. Тип\n";

cout<<"4. Кол-во копирования\n";

cout<<"5. Сменить знак\n";

cout<<"6. Представимо в Long?\n";

cin>> menu;

getchar();

if(menu>0 && menu<7)

for(i=0; i<countDecimal; i++)

{

cout<<"Элемент "<<numbers[2][i]+1<<": ";

switch(menu)

{

case 1: p[numbers[2][i]]->print(); break;

case 2: cout<<p[numbers[2][i]]->strLenght()<<"\n"; break;

case 3: cout<<p[numbers[2][i]]->getType()<<"\n"; break;

case 4: p[numbers[2][i]]->printCopyCount(); break;

case 5:

(static\_cast<Decimal\*>(p[numbers[2][i]])->makeNegative()).print();

break;

case 6:

cout<<(static\_cast<Decimal\*>(p[numbers[2][i]])->isLong())<<"\n";

break;

};

}

else

cout<< "Неправильный ввод\n";

break;

case 4: cout<<"Строка-идентификатор\n";

for(i=0; i<countIdentifier; i++)

{

cout<<"Элемент "<<numbers[1][i]+1<<": ";

p[numbers[1][i]]->print();

};

cout<<"Десятичное число\n";

for(i=0; i<countDecimal; i++)

{

cout<<"Элемент "<<numbers[2][i]+1<<": ";

p[numbers[2][i]]->print();

};

cout<<"Первый операнд. Введите номер элемента: ";

cin>> operand1;

getchar();

system("cls");

cout<<"Строка\n";

for(i=0; i<countString; i++)

{

cout<<"Элемент "<<numbers[0][i]+1<<": ";

p[numbers[0][i]]->print();

};

cout<<"Строка-идентификатор\n";

for(i=0; i<countIdentifier; i++)

{

cout<<"Элемент "<<numbers[1][i]+1<<": ";

p[numbers[1][i]]->print();

};

cout<<"Десятичное число\n";

for(i=0; i<countDecimal; i++)

{

cout<<"Элемент "<<numbers[2][i]+1<<": ";

p[numbers[2][i]]->print();

};

cout<<"Второй операнд. Введите номер элемента: ";

cin>> operand2;

getchar();

cout<<"Введите оператор: ";

cin>> operator1;

getchar();

system("cls");

/\*проверка ввода\*/

if(operand1<0 || operand1>count || operand2<0 || operand2>count )

{

cout<<"Неправильный ввод\n";

break;

};

/\*определяем типы операндов\*/

typeOperand1=p[--operand1]->getType();

typeOperand2 =p[--operand2]->getType();

if((typeOperand1 == 2 && (operator1 == '+' || operator1 == '\*')) ||

(operator1!='+' && operator1!='-' &&

operator1!='\*' && operator1!='='))

{

cout << "Оператор " <<

'"'<<operator1<<'"'<<

" не определен для операнда №1\n";

break;

};

p[operand1]->print();

cout<< operator1<<"\n";

p[operand2]->print();

cout<<"=\n";

/\*в зависимости от типов операндов и оператора выполняем работу\*/

if(typeOperand1 == 2)

{

if(typeOperand2 == 1)

switch(operator1)

{

case '-': (\*static\_cast<Identifier\*>(p[operand1]) -

\*p[operand2]).print(); break;

case '=': (\*static\_cast<Identifier\*>(p[operand1]) =

\*p[operand2]).print(); break;

};

if(typeOperand2 == 2)

switch(operator1)

{

case '-': (\*static\_cast<Identifier\*>(p[operand1]) -

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

case '=': (\*static\_cast<Identifier\*>(p[operand1]) =

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

};

if(typeOperand2 == 3)

switch(operator1)

{

case '-': (\*static\_cast<Identifier\*>(p[operand1]) -

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

case '=': (\*static\_cast<Identifier\*>(p[operand1]) =

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

};

}

else

{

if(typeOperand2 == 1)

switch(operator1)

{

case '+': (\*static\_cast<Decimal\*>(p[operand1]) +

\*p[operand2]).print(); break;

case '-': (\*static\_cast<Decimal\*>(p[operand1]) -

\*p[operand2]).print(); break;

case '\*': (\*static\_cast<Decimal\*>(p[operand1]) \*

\*p[operand2]).print(); break;

case '=': (\*static\_cast<Decimal\*>(p[operand1]) =

\*p[operand2]).print(); break;

};

if(typeOperand2 == 2)

switch(operator1)

{

case '+': (\*static\_cast<Decimal\*>(p[operand1]) +

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

case '-': (\*static\_cast<Decimal\*>(p[operand1]) -

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

case '\*': (\*static\_cast<Decimal\*>(p[operand1]) \*

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

case '=': (\*static\_cast<Decimal\*>(p[operand1]) =

\*static\_cast<Identifier\*>(p[operand2])).print(); break;

};

if(typeOperand2 == 3)

switch(operator1)

{

case '+': (\*static\_cast<Decimal\*>(p[operand1]) +

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

case '-': (\*static\_cast<Decimal\*>(p[operand1]) -

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

case '\*': (\*static\_cast<Decimal\*>(p[operand1]) \*

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

case '=': (\*static\_cast<Decimal\*>(p[operand1]) =

\*static\_cast<Decimal\*>(p[operand2])).print(); break;

};

};

break;

default: cout<< "Неправильный ввод"; break;

}; getchar(); break;

case 3 : break;

};

}while(menu1 !=3);

for(i=0; i<count; i++)

delete p[i];

delete [] p;

return 0;

}

**Результат работы программы**

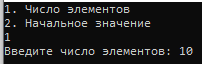
**1. Инициализация**



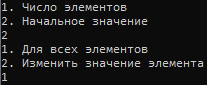
**2. Начальное значение**

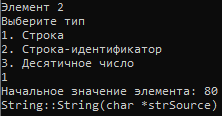
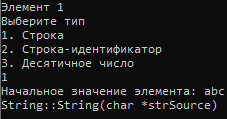


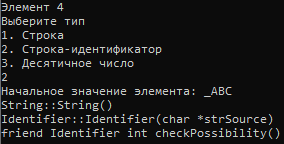
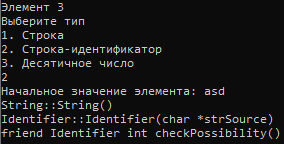
**1. Число элементов**

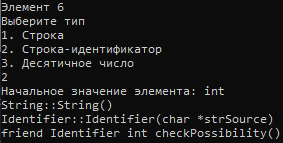
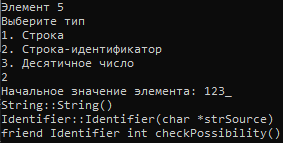


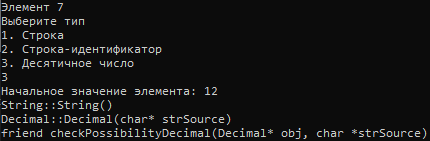
**2. Начальное значение**

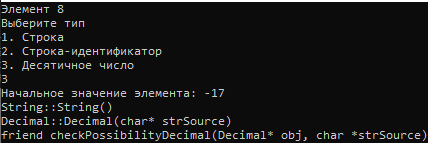


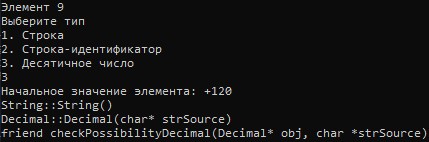


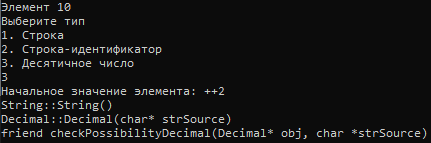






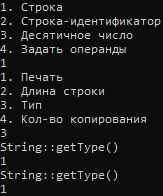
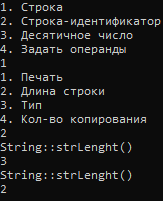
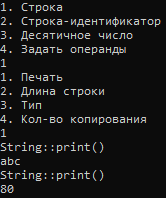




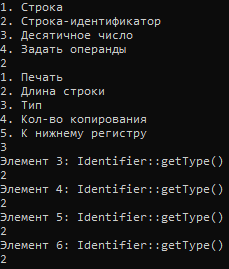
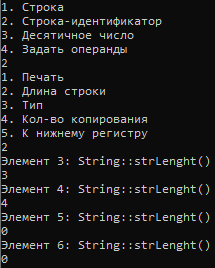
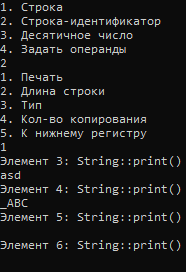


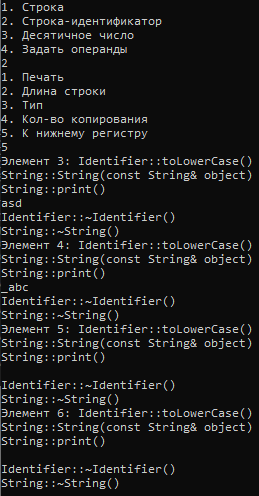
**2. Тестирование**

**1. Строка**

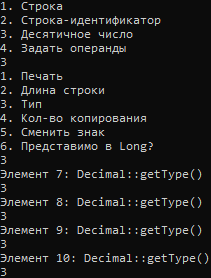
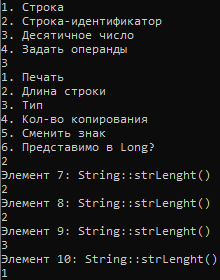
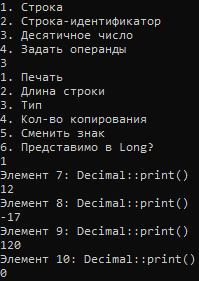


**2. Строка-идентификатор**

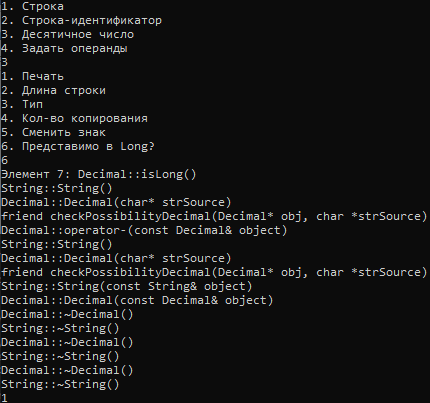


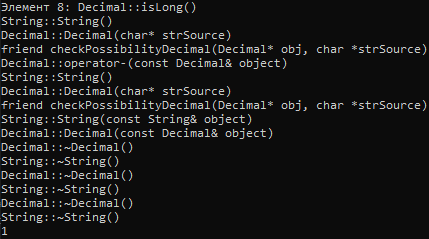


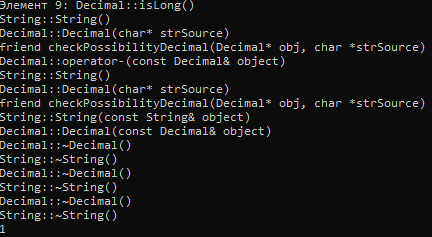
**3. Десятичное число**

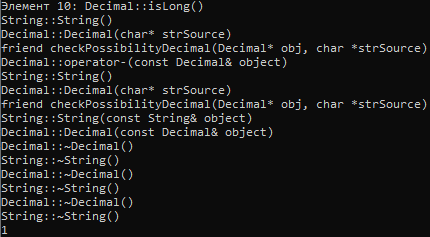


**Представимо в Long?**

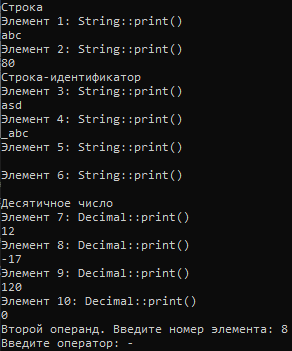
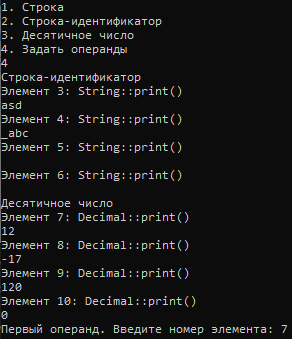


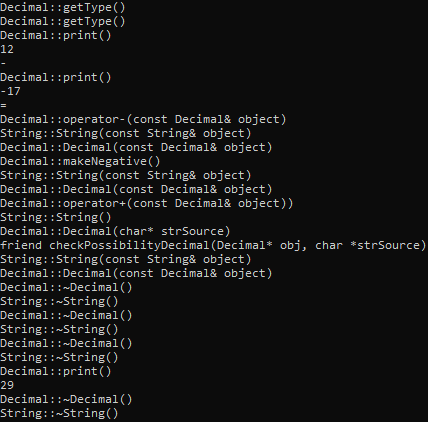


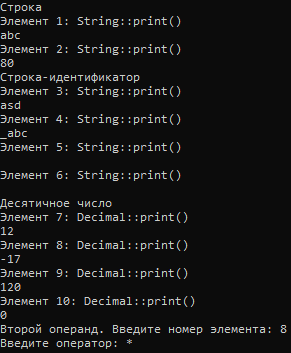
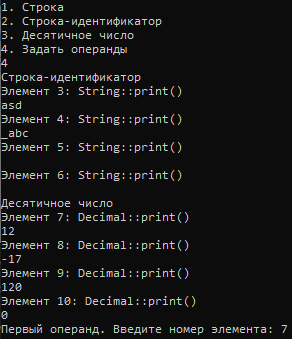


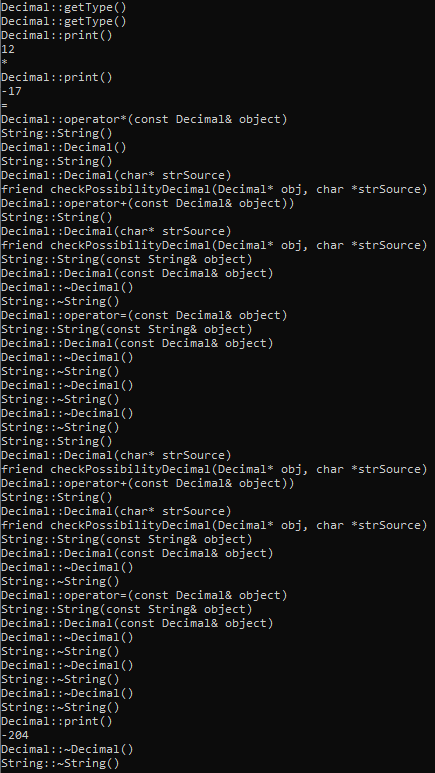


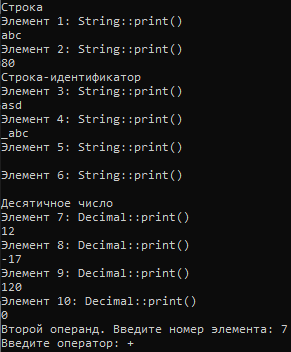
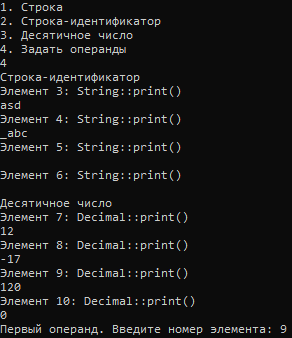
**4. Задать операнды**

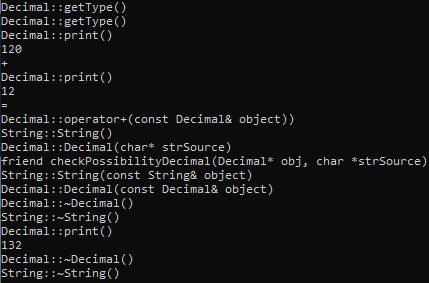


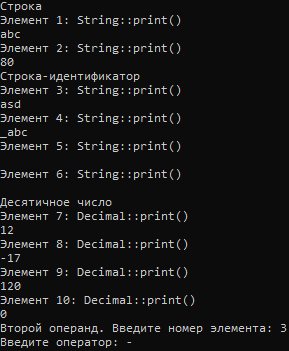
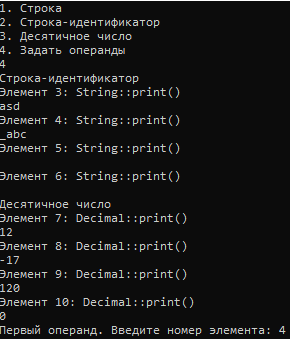


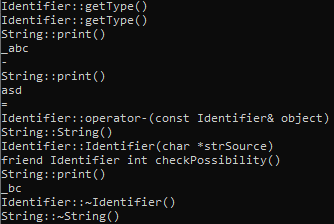


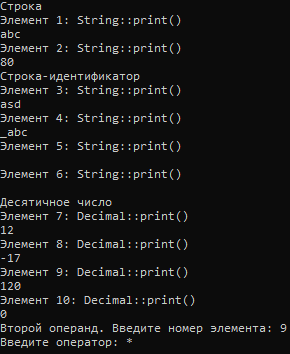
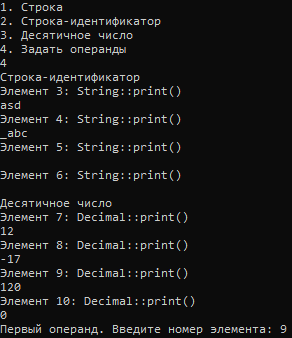


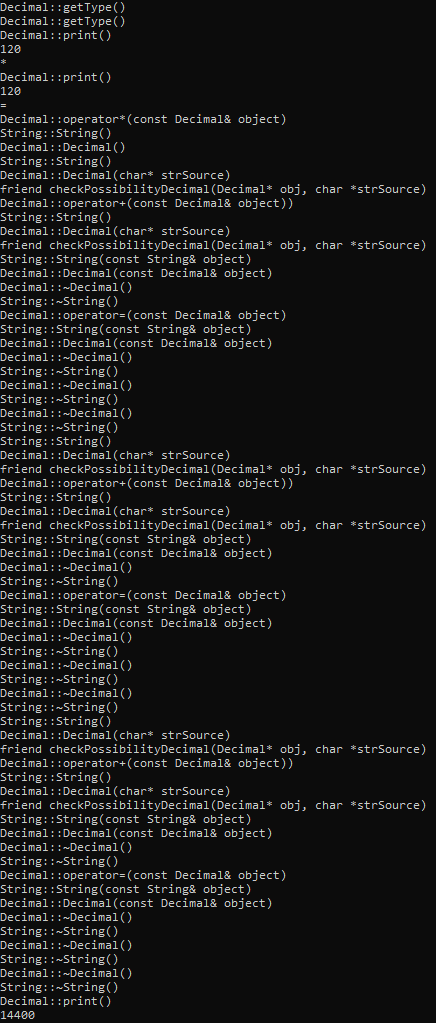












**3. Выход**

