МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №2**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Москвин Артём Артурович, группа М8О-208Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:** Разработать программу на языке C++ согласно варианту задания. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

**Вариант №15:**

Создать класс **TransNumber** для работы с трансцендентными числами. Трансцендентное число представлено парой (a, b), где a – действительная часть, b – трансцендентная часть. Трансцендентная часть представляет собой действительное число b, умноженное на константу. Реализовать арифметические операции (по аналогии с операциями над комплексными числами в алгебраической форме), и операции сравнения по значению (a + b ´).

**Описание программы**

Исходный код лежит в 3 файлах:

1. main.cpp - исполняемый код.

2. position.h - специальный файл .h, содержащий прототипы используемых мною функций.

3. position.cpp - реализация функций для моего задания.

4. CMakeLists.txt - специальный дополнительный файл типа CMakeLists.

**Дневник отладки:**

Во время выполнения данной лабораторной работы небольшие проблемы возникли с перегрузкой операторов, однако были почти сразу же устранены.

**Вывод:**  
 Данная лабораторная работа научила меня двум очень важным вещам: **1) перегрузке операторов.** Без перегрузки операторов не обходится ни один большой проект, это очень важное понятие в сфере объектно-ориентированного-программирования, ведь классы бывают совершенно разные, с разными полями. Например, в моем задании нужно складывать два объекта, хранящих в себе 3 поля: часы, минуты и секунды. Перегрузка операторов нам в этом деле очень сильно помогает.  
**2) пользовательским литералам.** Оказывается, это очень удобная и практичная вещь, о которой я никогда не знал. Прелесть данного средства в том, что мы вычисляем какие-то значения без использования вспомогательных функций, а попросту переопределением специального оператора.

**Исходный код**

position.h

#ifndef LAB0\_2\_POSITION\_H

#define LAB0\_2\_POSITION\_H

#include <cmath>

#include <iostream>

#include <string>

class Position{

private:

   int latitude, longitude; // широта и долгота

public:

    Position();

    Position(int latitude, int longitude);

    Position operator +(const Position &b);

    Position operator -(const Position &b);

    Position operator \*(const Position &b);

    Position operator /(const Position &b);

    void static Compare(Position a, Position b);

    friend std::istream& operator >>(std::istream& is, Position &pos);

    friend std::ostream& operator <<(std::ostream& os, Position &pos);

    friend bool operator ==(Position a, Position b);

};

Position operator "" \_pos(const char\* str, size\_t size);

#endif //LAB0\_2\_POSITION\_H

position.cpp   
  
#include "position.h"

Position::Position() {}

Position::Position(int latitude, int longitude){

    this->latitude = latitude;

    this->longitude = longitude;

}

Position Position::operator +(const Position &b){

    int newLatitude = latitude + b.latitude;

    int newLongitude = longitude + b.longitude;

    if(newLatitude > 90 || newLatitude < -90){

        newLatitude = 180 - abs(newLatitude);

    }

    if(newLongitude < -180){

        newLongitude = 360 + newLongitude;

    }

    else if(newLongitude > 180){

        newLongitude = newLongitude - 360;

    }

    return Position(newLatitude, newLongitude);

}

Position Position::operator -(const Position &b){

    int newLatitude = latitude - b.latitude;

    int newLongitude = longitude - b.longitude;

    if(newLatitude > 90 || newLatitude < -90){

        newLatitude = 180 - abs(newLatitude);

    }

    if(newLongitude < -180){

        newLongitude = 360 + newLongitude;

    }

    else if(newLongitude > 180){

        newLongitude = newLongitude - 360;

    }

    return Position(newLatitude, newLongitude);

}

Position Position::operator \*(const Position &b){

    int newLatitude = latitude \* b.latitude;

    int newLongitude = longitude \* b.longitude;

    if (newLatitude > 90){

        newLatitude %= 90;

    }

    else if (newLatitude < -90){

        newLatitude \*= -1;

        newLatitude %= 90;

        newLatitude \*= -1;

    }

    if (newLongitude > 180){

        newLongitude %= 180;

    }

    else if (newLongitude < -180){

        newLongitude \*= -1;

        newLongitude %= 180;

        newLongitude \*= -1;

    }

    return Position(newLatitude, newLongitude);

}

Position Position::operator /(const Position &b){

    int newLatitude = latitude / b.latitude;

    int newLongitude = longitude / b.longitude;

    return Position(newLatitude, newLongitude);

}

bool operator ==(Position a, Position b){

    return a.latitude == b.latitude && a.longitude == b.longitude;

}

void Position::Compare(Position a, Position b){

    if (a == b){

        std::cout << "Positions are equal\n";

    }

    else{

        if (a.latitude == b.latitude){

            std::cout << "Positions have the same latitude and ";

        }

        else if(a.latitude > b.latitude){

            std::cout << "First position is northern than second and ";

        }

        else{

            std::cout << "Second position is northern than first and ";

        }

        if (a.longitude == b.longitude){

            std::cout << "positions have the same longitude\n";

        }

        else if(a.longitude > b.longitude){

            std::cout << "first position is eastern than second\n";

        }

        else{

            std::cout << "second position is eastern than first\n";

        }

    }

}

std::istream& operator >>(std::istream& is, Position &pos){

    is >> pos.latitude >> pos.longitude;

    return is;

}

std::ostream& operator <<(std::ostream& os, Position &pos){

    os << "latitude, longitude: (" << pos.latitude << "," << pos.longitude << ")\n";

    return os;

}

Position operator "" \_pos(const char\* str, size\_t size) {

    std::string strLat, strLong;

    int i = 0;

    while(str[i] != ','){

        strLat.push\_back(str[i]);

        ++i;

    }

    ++i;

    while(str[i] != '\0'){

        strLong.push\_back(str[i]);

        ++i;

    }

    int latitude = std::stoi(strLat), longitude = std::stoi(strLong);

    return Position(latitude, longitude);

}

main.cpp  
  
#include "position.h"

int main() {

    Position pos1;

    Position pos2;

    Position pos3;

    std::cout << "Enter 1st coordinates (latitude, then longitude):\n";

    std::cin >> pos1;

    std::cout << "Enter 2nd coordinates (latitude, then longitude):\n";

    std::cin >> pos2;

    std::cout << "Enter 3d coordinates (latitude, then longitude):\n";

    std::cin >> pos3;

    Position::Compare(pos1, pos2);

    Position::Compare(pos1, pos3);

    Position sum = pos1 + pos3;

    Position diff = pos1 - pos3;

    Position mult = pos1 \* pos3;

    Position substr = pos1 / pos3;

    std::cout << sum;

    std::cout << diff;

    std::cout << mult;

    std::cout << substr;

    std::cout << "Test of literals\n";

    Position pos4 = "35,90"\_pos;

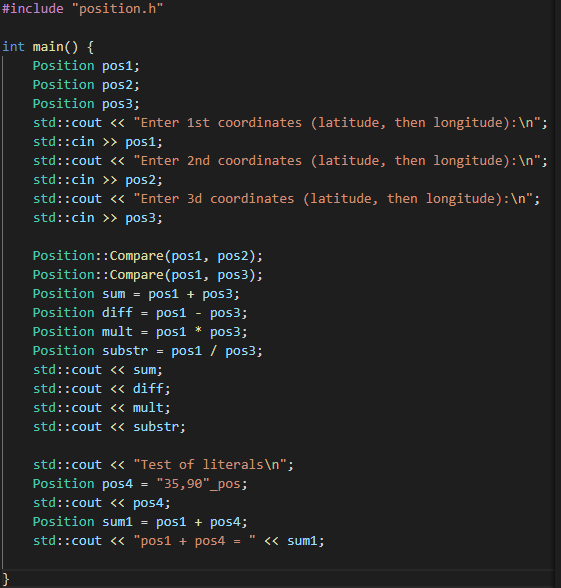
    std::cout << pos4;

    Position sum1 = pos1 + pos4;

    std::cout << "pos1 + pos4 = " << sum1;

}

**Пример работы:**

****

