МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №1 по курсу

объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Михеева Кристина Олеговна, группа М8О-207Б-20*
Преподаватель *Дорохов Евгений Павлович*

Условие.

Вариант 17: Создать класс Budget для работы с бюджетом. Класс состоит из двух вещественных чисел (a,b). Где a – собственная часть средств бюджета в рублях, b – заемная часть средств бюджета рублях. Оба числа должны округляться до второго знака после запятой. Реализовать арифметические операции сложения, вычитания, умножения и деления, а также операции сравнения.

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Описание программы .

Исходный код лежит в 3 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню
2. Budget.h: описание класса бюджета
3. Budget.cpp: описание класса точки

Дневник отладки:

В данной лабораторной критичных ошибок не было выявлено.

Выводы:

В данной лабораторной работе я познакомилась и научилась
базовым понятий ООП,с классами в C++. Ведь от любого метода программирования мы ждем, что он поможет нам в решении наших проблем. Но одной из самых значительных проблем в программировании является сложность. Чем больше и сложнее программа, тем важнее становится разбить ее на небольшие, четко очерченные части. Чтобы побороть сложность, мы должны абстрагироваться от мелких деталей. В этом смысле классы представляют собой весьма удобный инструмент.

- Классы позволяют проводить конструирование из полезных компонент, обладающих простыми инструментами, что дает возможность абстрагироваться от деталей реализации.

- Данные и операции вместе образуют определенную сущность и они не «размазываются» по всей программе, как это нередко бывает в случае процедурного программирования.
- Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения.
- Инкапсуляция информации защищает наиболее критичные данные от несанкционированного доступа.

ООП дает возможность создавать расширяемые системы. Это одно из самых значительных достоинств ООП и именно оно отличает данный подход от традиционных методов программирования. Расширяемость означает, что существующую систему можно заставить работать с новыми компонентами, причем без внесения в нее каких-либо изменений. Компоненты могут быть добавлены на этапе выполнения.

Листинг:

Budget.h

```
#ifndef BUDGET_H
#define BUDGET_H
#include <iostream>

class Budget {
    public:
        Budget();
        Budget(double a, double b);
        Budget(std::istream &is);
        void Rouding();
        void Difference();
        void Summa();
        void Multiplier();
        void Division();
        void Compare();
        void Print(std::ostream &os);
        ~Budget();

    private:
    double personal;
    double credit;

};
#endif
```

Budget.cpp

```
#include "Budget.h"
#include <cmath>
```

```
Budget::Budget() {
    personal = 0.0;
    credit = 0.0;
    std:: cout << "Start: " << std:: endl;
}

Budget::Budget(double a, double b) {
    if (a > 0.0 && b > 0.0){
        personal = a;
        credit = b;
    }
    else {
        std:: cout << "Please enter positive numbers!" << std:: endl;
    }
    std:: cout << "The budget according to your parameters has been created"
<< std:: endl;
}
Budget::Budget(std::istream &is) {
    std:: cout << "Please enter your budget data: " << std:: endl;
    is >>  personal >> credit;
    if ((personal <= 0.0) || (credit <= 0.0)) {
        std:: cout << "Invalind input. Enter again!" << std:: endl;
        is >> personal >> credit;
    }
    std:: cout << "The budget has been created via istream" << std:: endl;
}

void Budget::Rouding() {
    this->personal = round (this->personal * 100.0) / 100.0;
    this->credit = round (this->credit * 100.0) / 100.0;
    std:: cout << "Rouding: " << this->personal << " " << this->credit <<
std:: endl;
}

void Budget::Difference() {
    double differ = personal - credit;
    std:: cout << "Difference: " << differ << std:: endl;
}

void Budget::Summa() {
    double budget = personal + credit;
    std:: cout << "Total budget: " << budget << std:: endl;
}

void Budget::Multiplier(){
    double mult = personal * credit;
    std:: cout << "Multiplier: " << mult << std:: endl;
}

void Budget::Division(){
    double div = personal / credit;
    std:: cout << " Division: " << div << std:: endl;
}
```

```
void Budget::Compare(){
     if (personal > credit){
        std:: cout << "The personal is more that credit!" << std:: endl;
    }
    else if (personal == credit) {
        std:: cout << "Budget are equal!" << std:: endl;
    }
    else {
        std:: cout << "The credit is more that personal!" << std:: endl;
    }
}

void Budget::Print(std::ostream& os) {
    os << "Your budget is: " << personal << "," << credit << std:: endl;
}

Budget::~Budget() {
    std:: cout << "FROM DESTRUCTOR: Your budget has been deleted" << std::
endl;
}
```

main.cpp

```
##include "Budget.h"

int main(){
   Budget a(std:: cin);
   a.Rouding();
   a.Difference();
   a.Summa();
   a.Division();
   a.Multiplier();
   a.Compare();
   a.Print(std:: cout);
   return 0;

}
```