C++

Урок 17

Разбор ДЗ

Что было необычного?

Повторение

Дружественная ф-ия

Дружественные функции

Дружественные функции - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

Дружественные функции

Дружественные функции - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

```
friend тип результата имя функции(параметры) {}
```

Разбор дз

```
int main()
{
    drob a(7, 9), b(3, 5);
    drob c = 5 + a;
    drob d = a + 5;
    a.show();
    c.show();
    d.show();
}
```

Main

Разбор дз

```
friend drob operator+(const int &, const drob &); //дружественная функция
```

```
drob operator+(const int &test, const drob &ty)
{
    drob res(test * ty.zn + ty.ch, ty.zn);
    return res;
}
```

Реализация ф-ий

Дружественные функции

Дружественные функции - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

Дружественные функции

Дружественные функции - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

```
friend тип результата имя функции(параметры) {}
```

Перегрузка ввода и вывода

Перегрузка ввода и вывода

Перегрузка **<<** (вывода)

Чтобы выводить значения в объект класса через стандартный поток вывода **<<** требуется перегрузить данный оператор для данного класса.

Перегрузить можно оператор вывода в **поток ostream**, что позволит использовать экземпляры класса в строке потока вывода с таким же синтаксисом, как и для встроенных типов.

При этом в качестве **левого операнда** оператор << должен иметь **неконстантную ссылку** на поток ostream, а правым **константная ссылка на объект** => данный оператор всегда определяется **внешней функцией**, а не методом класса.

В качестве результата следует возвращать **исходную ссылку** на поток.

Перегрузка ввода и вывода

Перегрузка **>>** (ввода)

Первый параметр оператора >> представляет **ссылку на объект istream**, с которого осуществляется чтение. Второй параметр представляет **ссылку на неконстантный объект**, в который надо считать данные.

В качестве результата операторы возвращают ссылку на поток ввода istream из первого параметра.

Абстракция и интерфейс

```
istream &operator>>(istream &in, drob &test)
{
    in >> test.ch >> test.zn;
    return in;
}

ostream &operator<<(ostream &out, const drob &test)
{
    out << test.ch << "/" << test.zn;
    return out;
}
```

Листинг кода

Абстракция и интерфейс

Абстракция данных – это парадигма проектирования, основанная на разделении **интерфейса** и **реализации**. То есть **абстрактный класс** может содержать чисто виртуальные функции и сами функции.

Интерфейс – абстрактный класс, у которого все методы являются чисто виртуальными.

Абстракция и интерфейс

```
class Person
{
private:
    string name;
    int *age;
    string email;

    void read()
    {
        cout << "Введите email адресс " << endl;
        cin >> email;
        valid();
    }
    void valid()
    {
        auto begin = email.begin(), end = email.end();
        if ((begin = find(begin, end, '@')) != end &&
            find(begin, end, '.') != end)
        {
            cout << "Email accepted" << std::endl;
        }
        else
        {
            cout << "Email rejected" << std::endl;
            read();
        }
    }

public:
    Person(string name = " ", int age = 0, string email = " ") : name(name), email(email)
    {
        this->age = new int(age);
        if (name == " ")
            name = "Имя не введено";
        valid();
    }
    virtual void info() = 0;
    virtual ~Person()
    {
        delete age;
    }
};
```

Абстрактный класс

Абстракция и интерфейс

```
class person
{
protected:
    string name;
    int *age;
    string email;

public:
    virtual void read() = 0;
    virtual void info() = 0;
    virtual ~person() = 0;
};

person::~person(){};
```

Интерфейс

Виртуальный деструктор

Виртуальный деструктор

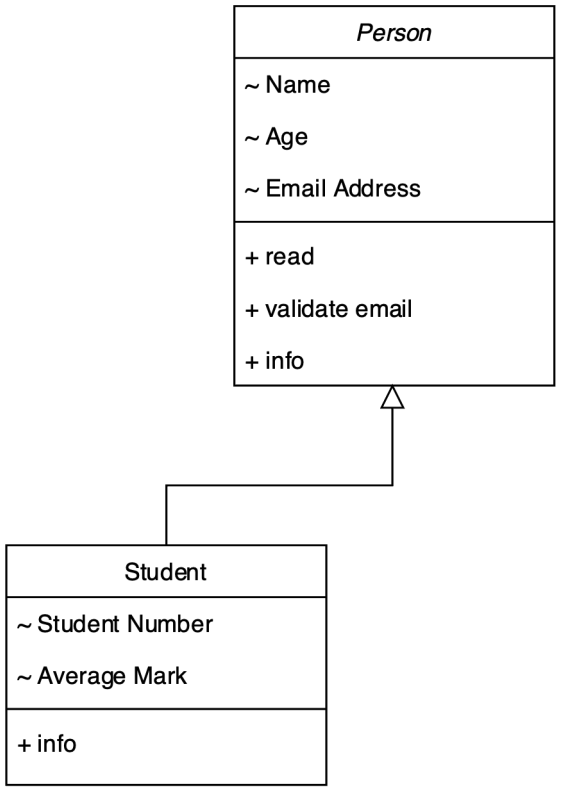
Деструктор — это функция-член, которая вызывается автоматически при выходе объекта из области действия или явно уничтожена вызовом delete .

Деструктор имеет то же имя, что и класс, перед которым предшествует тильда (~).

Виртуальный деструктор нужен для очищения памяти у наследников, если они создавались через указатель базового класса.

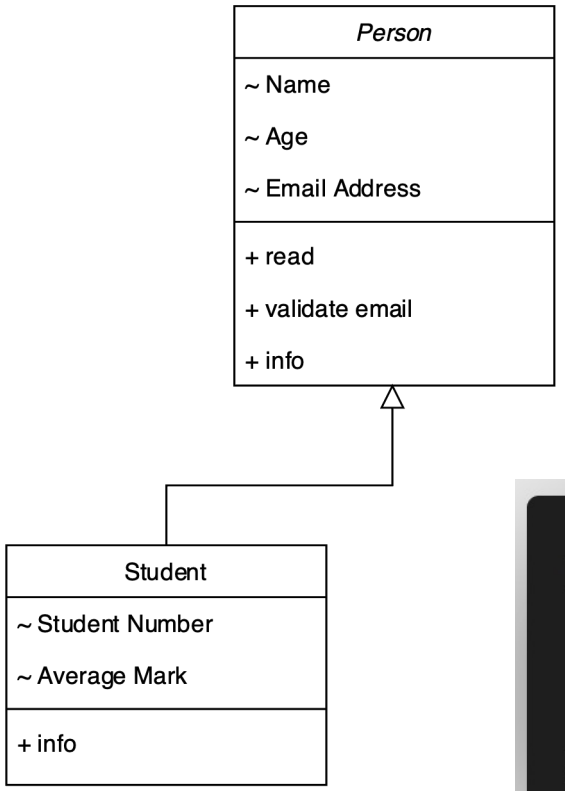
Задача

Задача: Реализуйте класс Student по диаграмме на след странице.



Особенности:

- Student Number – целое число, которое > 0
- Average Mark – вещественное число > 0
- info - функция вывода полей



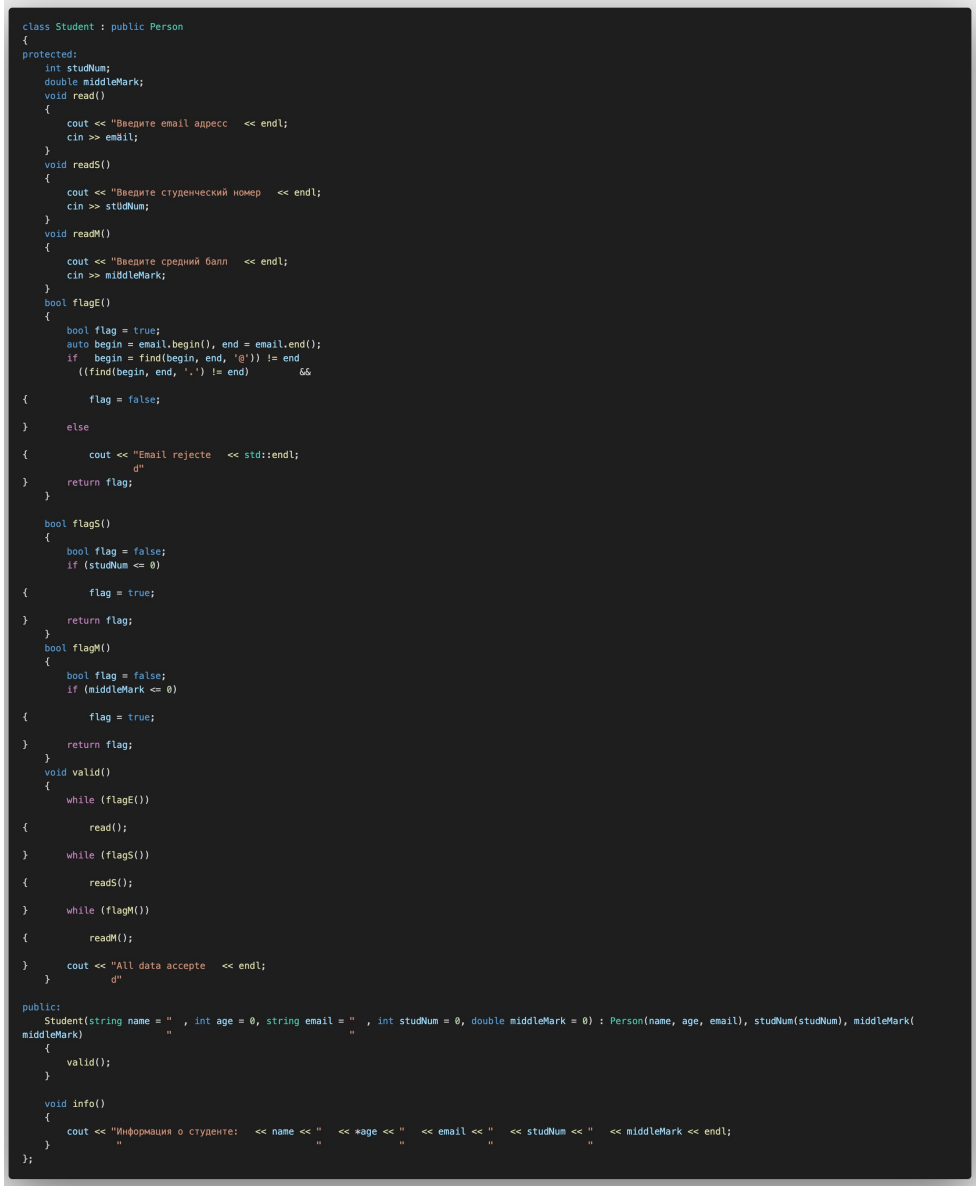
Особенности:

- Student Number – целое число, которое > 0
- Average Mark – вещественное число > 0
- info - функция вывода полей

```
int main()
{
    Person *Melman = new Student;
    Melman->info();
    delete Melman;
}
```

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

class Person
{
protected:
    string name;
    int *age;
    string email;

    virtual void read()
    {
        cout << "Введите email адресс " << endl;
        cin >> email;
        valid();
    }
    virtual void valid()
    {
        auto begin = email.begin(), end = email.end();
        if ((begin = find(begin, end, '@')) != end &&
            find(begin, end, '.') != end)
        {
            cout << "Email accepted" << std::endl;
        }
        else
        {
            cout << "Email rejected" << std::endl;
            read();
        }
    }

public:
    Person(string name = " ", int age = 0, string email = " ") : name(name), email(email), age(new int(age))
    {
        // this->age = new int(age);
        if (name == " ")
            name = "Имя не введено";
        valid();
    }
    Person(const Person &test)
    {
        this->name = name;              //копируем значение поля
        this->email = email;            //копируем значение поля
        this->age = new int(*test.age); //выделили память под переменную и занесли значение
        cout << "конструктор копирования" << endl;
    }

    virtual void info()
    {
        cout << "Информация о человеке: " << name << " " << *age << " " << email << endl;
    }
    ~Person()
    {
        delete age;
    }
};
```

Класс Person



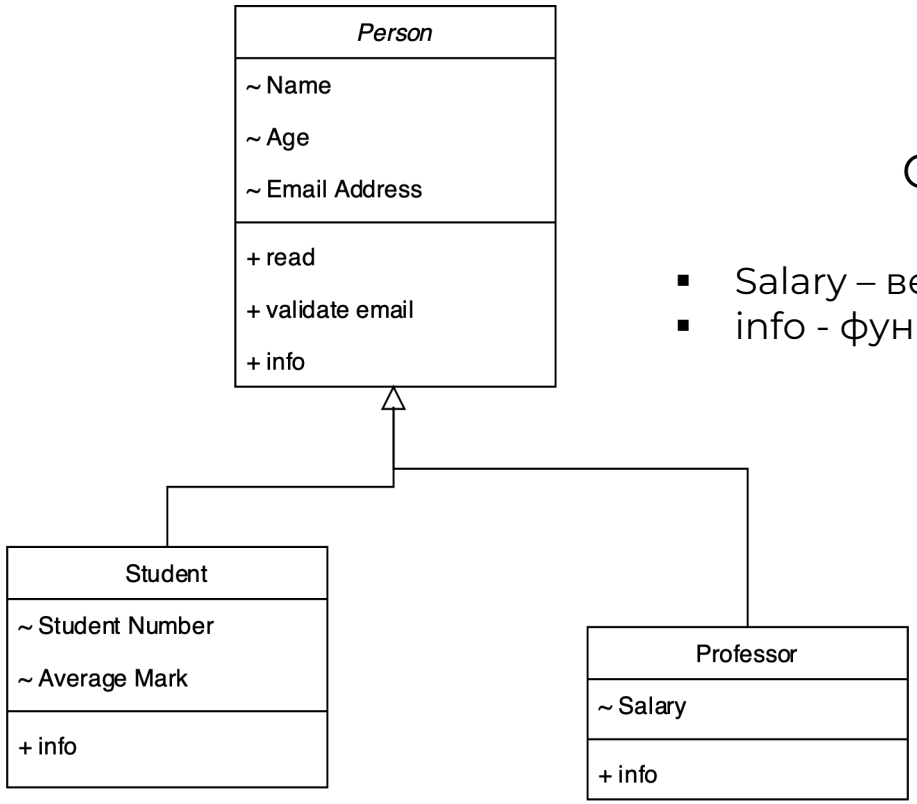
```
class Student : public Person
{
protected:
    int studNum;
    double middleMark;
    void read()
    {
        cout << "Введите email адресс   << endl;
        cin >> email;
    }
    void readS()
    {
        cout << "Введите студенческий номер   << endl;
        cin >> studNum;
    }
    void readM()
    {
        cout << "Введите средний балл   << endl;
        cin >> middleMark;
    }
    bool flagE()
    {
        bool flag = true;
        auto begin = email.begin(), end = email.end();
        if   begin = find(begin, end, '@')) != end
          ((find(begin, end, '.') != end)         &&

{           flag = false;

}       else

{           cout << "Email rejecte   << std::endl;
                    d"
}       return flag;
    }

    bool flagS()
    {
        bool flag = false;
        if (studNum <= 0)

{           flag = true;

}       return flag;
    }
    bool flagM()
    {
        bool flag = false;
        if (middleMark <= 0)

{           flag = true;

}       return flag;
    }
    void valid()
    {
        while (flagE())

{           read();

}       while (flagS())

{           readS();

}       while (flagM())

{           readM();

}       cout << "All data accepte   << endl;
    }           d"

public:
    Student(string name = "  , int age = 0, string email = "  , int studNum = 0, double middleMark = 0) : Person(name, age, email), studNum(studNum), middleMark(
middleMark)                 "                                   "
    {
        valid();
    }

    void info()
    {
        cout << "Информация о студенте:   << name << "   << age << "   << email << "   << studNum << "   << middleMark << endl;
    }            "                                  "             "               "                "
};
```

Класс Student

Задача

Задача: Реализуйте класс Professor по диаграмме на след странице.

В main реализуйте вектор указателей на объекты и вводите пока не введено сообщение “exit”.



Особенности:

- Salary – вещественное число, которое > 0
- info - функция вывода полей

Задача

Задача: напишите ф-ию count, которая считает кол-во студентов и преподавателей.