Класове

Предишният път започнахме с класове, сега ще довършим.

Ако класът е тип данна, то променливите се наричат обекти или инстации.

Пример: `Rational a` - `Rational` е класът, `a` е обект от тип `Rational`

Методите трябва да бъдат вътре в класовете

Нека разгледаме примерът от `Rational` и по-конкретно, `add()`. В момента той е извън класът `Rational`, което налага използване на методите за достъп до член даннитеclass Person { Person name; }. Ако преместим `add()` вътре в класът, избягваме нуждата от това.

```
#include <iostream>

class Rational
{
    private:
    int numerator;
    int denominator;

    public:
    void setNumerator(int n)
    {
        numerator = n;
    }
    int getNumerator()
    {
        return numerator;
    }
    void setDenominator(int n)
    {
        numerator = n;
    }
    int getDenominator()
    {
        return denominator;
    }

    Rational add(Rational& b)
    {
        Rational c;
        if(this->denominator == b.denominator)
        {
            c.numerator = this->numerator + b.denominator;
            c.denominator =  this->denominator;
        }
        else
```

```
        {
            c.numerator = this->numerator*b.denominator + b.numerator*this->denominator;
            c.denominator = this->denominator*b.denominator;
        }

        return c;
    }
};


int main()
{
    Rational a,b,c;

    a.setNumerator(5);
    a.setDenominator(7);

    b.setNumerator(3);
    b.setDenominator(4);

    c = a.add(b);

    return 0;
}
```

Забелязваме няколко неща:

За да имаме достъп до член-данните на класът, пред името на метода трябва да се напише името на класът, последвано от `::`

Пример, с метод за извеждане на рационално число:

```
...
void Rational::print()
{
    std::cout << this->numerator << "/" << this->denominator;
}
...
```

`add()` приема само един аргумент

Когато метод е вътре в клас, то този метод е "закрепен" за текущита инстанция (обект), т.е. обекта се подава като невидим параметър на метода.

Какво е `this` ?

this е указател към текущият обект

Правила за писане на класове

Обекти на класът не могат да бъдат в член-данните на класът

```
class Person
{
    Person name;
    int age;
};
```

Това е грешно

```
class Person
{
    Person* name;
    int age;
};
```

Това е правилно

Това е поради фактът, че при първият пример, размерът на `Person` ще бъде размерът на `Person` + размерът на `int`. (Което довежда до невъзможност за изчисляване на размерът)

Във вторият пример, `name` е тип указател, което е с фиксиран размер (4b/8b), което ни дава фиксиран размер на `Person`

Разделяне на декларациите на клас и дефинициите

С цел по-лесното използване, декларацията на един клас се поставя в .h файл, а самите дефиниции в .cpp файлове

Пример за .h файл

```
class Rational
{
    private:
    int numerator;
    int denominator;

    public:
    void setNumerator(int n);
    void setDenominator(int n);
    int getNumerator();
    int getDenominator();

    Rational add(const Rational& b);
};
```

Пример за .cpp файл

```
#include "Rational.h"
void Rational::setNumerator(int n)
{
    numerator = n;
```

```
}
int Rational::getNumerator()
{
    return numerator;
}
void Rational::setDenominator(int n)
{
    numerator = n;
}
int Rational::getDenominator()
{
    return denominator;
}

Rational Rational::add(const Rational& b)
{
    Rational c;
    if(this->denominator == b.denominator)
    {
        c.numerator = this->numerator + b.denominator;
        c.denominator =  this->denominator;
    }
    else
    {
        c.numerator = this->numerator*b.denominator + b.numerator*this->denominator;
        c.denominator = this->denominator*b.denominator;
    }
    return c;
}
```

Използване на `const`

Нека разгледаме методът `getNumerator()`. Ако трябваше да го разпишем извън класът, то той би изглеждал по следният начин:

```
int getNumerator(Rational& a)
{
    return a.numerator;
}
```

Добрият стил за писане на код изисква аргументите на функция, която не променя обектите да са константни. (т.е. вместо `Rational& a`, да имаме `const Rational& a`)

Когато обаче методите са вътре в класът, текущият обект с когото работим, се подава неявно. Как да напишем `const` тогава - като добавим `const` след декларацията на функцията

```
// .h file
int Rational::getNumerator() const
...

// .cpp file
int Rational::getNumerator() const
{
    return numerator;
}
```

Задачи за упражнение

Задача 1:

Да се напише клас `Point`, който да представя точка в двумерното пространство (x,y). Да се реализират методи за въвеждане координатите на точката от клавиатурата (x и y са на нов ред), принтиране на координатите на точката (формат - *(x,y)*) и метод, който изчислява разстоянието между две точки. Да се напише клас `Triange`, който е изграден от три точки, като има следните методи - изграждане на триъгълник по три точки, валидация дали е валиден триъгълник и извеждане на това какъв е триъгълника (равностранен, равнобедрен, разностранен).

Point.h

```
#pragma once

class Point
{
private:
    double x;
    double y;

public:
    void setXandY(); //Задаваме x и y от клавиатурата
    void print() const; //Принтираме точката
    double getX() const; //Взимаме x
    double getY() const; //Взимаме y
    double getDist(const Point& p) const; //Намираме разстоянието между две точки
};
```

Point.cpp

```
#include "Point.h"
#include <iostream>
#include <cmath>
void Point::setXandY()
{
    std::cin >> x >> y;
}
```

```
void Point::print() const
{
    std::cout << "(" << x << "," << y << ")";
}
double Point::getX() const { return x; }
double Point::getY() const { return y; }
double Point::getDist(const Point& p) const
{
    double result;

    result = sqrt(
        pow(x- p.x, 2) + pow(y-p.y, 2) //Формула за намиране на разстояние между две точки
в 2d пространство
    );

    return result;
}
```

Triangle.h

```
#pragma once
#include "Point.h"
class Triangle
{
private:
    double sideA;
    double sideB;
    double sideC;

    Point a;
    Point b;
    Point c;

public:
    void setPoints(const Point& a, const Point& b, const Point& c);
    void typeOfTriangle() const;
    bool validTriangle() const;
};
```

Triangle.cpp

```
#include <iostream>
#include "Triangle.h"
void Triangle::setPoints(const Point& a, const Point& b, const Point& c)
{
    this->a = a;
    this->b = b;
```

```
    this->c = c;
    this->sideA = a.getDist(b);
    this->sideB = b.getDist(c);
    this->sideC = c.getDist(a);
}
void Triangle::typeOfTriangle() const
{
    //Проверка за страните
    if(sideA == sideB && sideB == sideC)
    {
        std::cout << "Equal sided";
    }
    else if((sideA == sideB && sideB != sideC) || (sideC == sideB && sideB != sideA) ||
(sideA == sideC && sideB != sideC))
    {
        std::cout << "Two sides are equal";
    }
    else
    {
        std::cout << "No equal sides";
    }
}
bool Triangle::validTriangle() const
{
    /*
    Проверка за валиден триъгълник:
        Сборът на всеки две страни трябва да е по-голям от третата
    */
    return (sideA + sideB > sideC && sideA + sideC > sideB && sideB + sideC > sideA);
}
```

Задача 2:

Да се напише клас `Date`, който да съдържа в себе си денят, датата, месец и годината. Освен това, да има метод за въвеждане на дата, и извеждане в следните формати - dd/mm/yyyy, dd.month_name.year, day dd month_name.

Date.h

```
class Date
{
    private:
        int day;
        int month;
        int year;
        char* dayName;

    public:
        void setDay(int);
        void setMonth(int);
        void setYear(int);
        void setDayName(char*);
```

```
        void displayA() const; //dd/mm/yyyy
        void displayB() const; //dd.month_name.year
        void displayC() const; //day dd month_name
};
```

Date.cpp

```
#include "Date.h"
#include <iostream>

void Date::setDay(int d)
{
    day = d;
}
void Date::setMonth(int d)
{
    month = d;
}
void Date::setYear(int d)
{
    year = d;
}
void Date::setDayName(char* n)
{
    dayName = n;
}

void Date::displayA() const //dd/mm/yyyy
{
    std::cout << day << "/" << month << "/" << year;
}
void Date::displayB() const //dd.month_name.year
{
    std::cout << day << "/";

    if(month == 1) std::cout << "January";
    else if(month == 2) std::cout << "February";
    else if(month == 3) std::cout << "March";
    else if(month == 4) std::cout << "April";
    else if(month == 5) std::cout << "May";
    else if(month == 6) std::cout << "June";
    else if(month == 7) std::cout << "July";
    else if(month == 8) std::cout << "August";
    else if(month == 9) std::cout << "September";
    else if(month == 10) std::cout << "Octomber";
    else if(month == 11) std::cout << "November";
    else if(month == 12) std::cout << "December";

    std::cout << "/" << year;
}
```

```cpp
void Date::displayC() const //day dd month_name
{
    int i = 0;
    while(dayName[i] != '\0')
    {
        std::cout << dayName[i];
        i++;
    }

    std::cout << " " << day << " ";

    if(month == 1) std::cout << "January";
    else if(month == 2) std::cout << "February";
    else if(month == 3) std::cout << "March";
    else if(month == 4) std::cout << "April";
    else if(month == 5) std::cout << "May";
    else if(month == 6) std::cout << "June";
    else if(month == 7) std::cout << "July";
    else if(month == 8) std::cout << "August";
    else if(month == 9) std::cout << "September";
    else if(month == 10) std::cout << "Octomber";
    else if(month == 11) std::cout << "November";
    else if(month == 12) std::cout << "December";
}
```