

Конструктори – constructor, copy constructor

Конструктори

Понякога бихме искали да зададем начални стойности на член-данните на класовете, или да се изпълни някакъв код/метод по време на създаването на обектите.

Нека разгледаме примерът с `Rational`

```
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)
};
```

С цел да избегнем създаването на случайни дробни, нека искаме всеки обект от тип `Rational` да има за числител 0 и знаменател 1

При създаването на обекти, се извиква специален метод, наречен **конструктор**.

Някои особености на **конструкторите** са:

- Името на конструктора съвпада с името на класа
- Типът на резултата не се указва
- Изпълнява се автоматично при създаване на обекта
- Един клас може да няма **явно** дефиниран конструктор (т.е. винаги има конструктор по подразбиране, дори и да не е дефиниран в класът), също така може да има повече от един

Нека напишем конструктор за класът `Rational`

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

    public:
        Rational(); //Конструктор
```

```

    void setNumerator(int n)
    void setDenominator(int n)
    int getNumerator()
    int getDenominator()

    Rational add(const Rational& b)
};

```

```

...
Rational::Rational()
{
    numerator = 0;
    denominator = 1;
}
...

```

Така, като създадем обект от тип `Rational`, той винаги по подразбиране ще има стойност 0 и 1

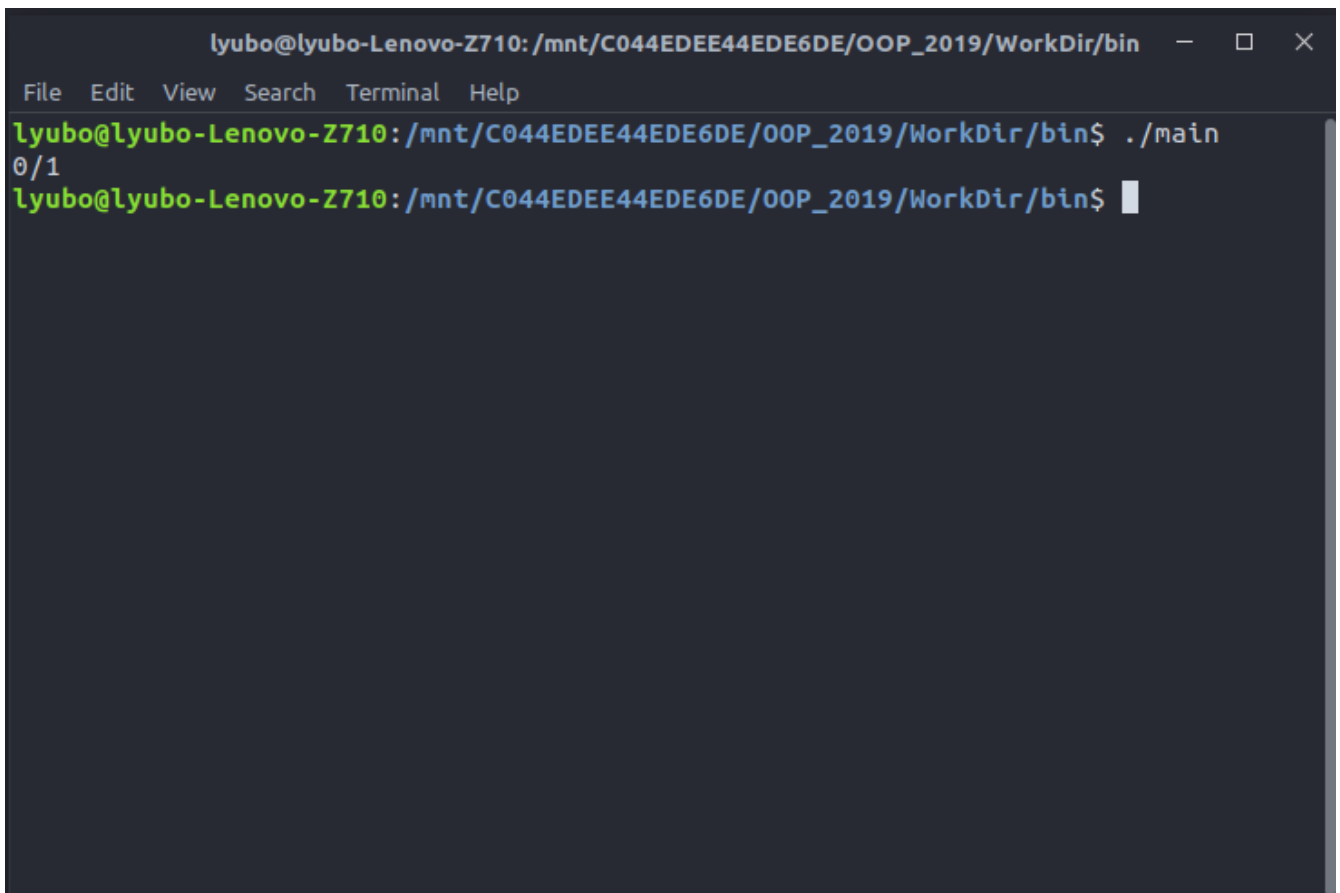
```

Rational a;
std::cout << a.getNumerator() << "/" << a.getDenominator();

//Друг начин за извикване на конструктор
//Rational a = Rational();

```

Съответният изход:



```

lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin
File Edit View Search Terminal Help
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$ ./main
0/1
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$

```

Конструкторът, който не приема никакви аргументи, се нарича **конструктор по подразбиране** (default-ен конструктор)

Конструктор с параметри

Също както функциите, конструкторите могат да се предифинират (конструкторите са частен случай на функциите)

Ако искаше да използваме конструктора да дадем някакви конкретни начални стойности, можем да предефинираме конструктора по подразбиране, като той да приема някакви стойности. В `Rational`, той ще изглежда по следният начин:

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

    public:
        Rational(); //Конструктор
        Rational(int, int); //Конструктор с два параметъра
        void setNumerator(int n)
        void setDenominator(int n)
        int getNumerator()
        int getDenominator()

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

```
...
Rational::Rational(int a, int b)
{
    numerator = a;
    denominator = b;
}
...
```

Нека отбележим, че в декларацията на конструктора не е необходимо да се дават имена на променливите. Това може да се направи в дефиницията на конструктора.

```
Rational a = Rational(3,5);

std::cout << a.getNumerator() << "/" << a.getDenominator() << std::endl;
```

```
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin
File Edit View Search Terminal Help
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$ ./main
3/5
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$
```

Когато използваме конструктор с параметри с цел да инициализираме член-данните на класа, можем да използваме следният синтаксис:

```
Rational::Rational(int a, int b): numerator(a), denominator(b) {}
```

Какъв е смисълът от това ?

- По-бърза работа на конструктора - инициализацията на променливите се случва преди тялото на конструктора
- Когато има член-данни, които са обекти, техните конструктори по подразбиране се извикват преди тялото на конструктора - т.е. ако направим следното:

```
A()
{
    obj = B(5);
}
```

То конструкторът на obj (което е инстанция на B) ще се извика два пъти - един път ще се извика конструкторът по подразбиране, а втори път, този който приема параметри. Но ако направим следното:

```
A(): B(5)
{
}
```

,то ще се извика само конструкторът с параметри на B

Конструктори с подразбиращи се параметри

Както функциите могат да имат подразбиращи се параметри, така могат и конструкторите

Пример за функция с подразбиращи се параметри

```
int foo(int a = 5, int b = 3)
{
    return a+b;
}
```

Тази функция може да се използва по следният начин:

```
foo();    //8,  защото a = 5, b = 3
foo(3);   //6,  защото a = 3, b = 3
foo(4,7); //11, защото a = 4, b = 7
```

По същият начин може да се напишат и конструктори на `Rational`

```
Rational::Rational(int a = 0, int b = 1):numerator(a), denominator(b) {}
```

Този конструктор може да изпълнява ролята на **конструктор по подразбиране**, защото можем да не подадем никакви аргументи на този конструктор, и тогава ще се използват стойностите по подразбиране.

Конструктор за копиране

Доста често се налага да създадем обект, който е копие на вече съществуващ. Това може да се случи по два начина:

```
Rational a(1,2);
Rational b;
b = a;
```

или

```
Rational a(1,2);
Rational b(a);
```

При първия вариант се използва **оператор =**, а във вторият **конструктор за копиране**. Сега ще разгледаме **конструктора за копиране**

За да се дефинира конструктор за копиране, използваме следният синтаксис

```
ClassName (const ClassName& objToCopyFrom)
{
    //Тяло на конструктора
}
```

или в примера за `Rational`:

```
...
Rational(const Rational&); //Конструктор за копиране
...
```

```
...
Rational::Rational(const Rational& rhs)
{
    numerator = rhs.numerator;
    denominator = rhs.denominator;
}
...
```

В този случай, подаваме на **copy-constructor-a** обекта от който ще копираме, като не забравяме `const` и `&`.

Някои особености на **конструктора за копиране**:

- Ако не е дефиниран явно, компилатора дефинира такъв по подразбиране
- Съществува само един **конструктор за копиране**
- Той винаги приема като аргумент обект от същият тип

```
#pragma once
class Rational
{
private:
    int numerator;
    int denominator;

public:
    Rational(int, int); //Конструктор с два параметъра, с аргументи по подразбиране
    Rational(const Rational&); //Конструктор за копиране
    void setNum(int a);
    void setDenom(int a);
    void print() const;

    int getNum() const;
    int getDenom() const;

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

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

Rational::Rational(int a = 0, int b = 1):numerator(a), denominator(b) {}
```

```
Rational::Rational(const Rational& rhs)
{
    numerator = rhs.numerator;
    denominator = rhs.denominator;
}

void Rational::setNum(int a) { numerator = a; }
void Rational::setDenom(int a) { denominator = a; }
void Rational::print() const
{
    std::cout << numerator << "/" << denominator << std::endl;
}

int Rational::getNum() const { return numerator; }
int Rational::getDenom()const { return denominator; }

Rational Rational::add(const Rational& b)
{
    Rational c;

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

    return c;
}
```

```
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin — □ ×
File Edit View Search Terminal Help
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$ ./main
12/9
lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$
```

Задачи за упражнение:

Задача 1:

Напишете клас `Login`, който съдържа потребителско име (`char[]` с динамичен размер), парола (`char[]` отново динамичен размер) и тип на сесията (`int`).

Напишете следните конструктори:

- Конструктор по подразбиране
- Конструктор, който приема един аргумент - потребителско име. При негово извикване, потребителят да въведе парола. Сесията да е равна на 1
- Конструктор, който приема два аргумента - потребителско име и парола. Сесията да е равна на 1
- Конструктор с три аргумента
- Конструктор за копиране

Login.h

```
class Login
{
    private:
        char* username;
        char* password;
        int sessionType;

    public:
        Login();
```



```

    Login(char*);
    Login(char*, char*);
    Login(char*, char*, int);
    Login(const Login&);

    void setUsername(const char*);
    void setPassword(const char*);
    void setSession(const int);

    void print() const;
    char* getUsername() const;
    char* getPassword() const;
    int getSession() const;
};

```

Login.cpp

```

#include <iostream>
#include "Login.h"
#include <cstring>

Login::Login(): sessionType(0)
{
    username = new char[5];
    strcpy(username, "1234");

    password = new char[5];
    strcpy(password, "1234");
}

Login::Login(char* user):sessionType(1)
{
    int size = strlen(user);
    username = new char[size+1];

    strcpy(this->username, user);

    int t;

    std::cout << "Password length: ";
    std::cin >> t;
    std::cout << std::endl;

    password = new char[t+1]; //За да има място и за \0

    std::cin>> password;
}

Login::Login(char* user, char* pass):sessionType(1)
{
    int size = strlen(user);
    username = new char[size];

    size = strlen(pass);
    password = new char[size];
}

```

```

        strcpy(this->username, user);
        strcpy(this->password, pass);
    }
Login::Login(char* user, char* pass, int session): sessionType(session)
{
    int size = strlen(user);
    username = new char[size];

    size = strlen(pass);
    password = new char[size];

    strcpy(this->username, user);
    strcpy(this->password, pass);
}
Login::Login(const Login& rhs)
{
    int size = strlen(username);
    username = new char[size];

    size = strlen(password);
    password = new char[size];

    strcpy(this->username, rhs.username);
    strcpy(this->password, rhs.password);
    sessionType = rhs.sessionType;
}

void Login::setUsername(const char* user)
{
    int size = strlen(user);
    username = new char[size];

    strcpy(this->username, user);
}
void Login::setPassword(const char* pass)
{
    int size = strlen(pass);
    password = new char[size];

    strcpy(this->password, pass);
}
void Login::setSession(const int session)
{
    sessionType = session;
}

void Login::print() const
{
    std::cout << username << " " << password << " " << sessionType << std::endl;
}
char* Login::getUsername() const

```

```
{
    return username;
}
char* Login::getPassword() const
{
    return password;
}
int Login::getSession() const
{
    return sessionType;
}
```
