4-MA’RUZA.

KONSTRUKTORLAR VA DESTRUKTORLAR. A’ZO O’ZGARUVCHILARNI INITSIALIZATSIYALASH.

SINF A'ZOLARINING STATIK O‘ZGARUVCHILARI

KONSTRUKTORLAR VA DESTRUKTORLAR

Agar sinfning (yoki strukturaning) barcha a'zolari ochiq bo‘lsa, biz sinfni (yoki strukturani) to‘g‘ridan-to‘g‘ri boshlang‘ich ro‘yxati yoki bir xil initsializatsiya (C ++ 11 da) yordamida initsializatsiyalashimiz mumkin:

```
class Boo
{
public:
    int m_a;
    int m_b;
};

int main()
{
    Boo boo1 = { 7, 8 }; // initsializatorlar ro‘yxati
    Boo boo2 { 9, 10 }; // uniform-initsializatsiya (C++11)

    return 0;
}
```

Xususiy a'zo o‘zgaruvchilari bo‘lgan sinfni *konstruktorlardan* foydalangan holda initsializatsiyalash mumkin.

Konstruktor - bu bir xil sinfdagi obyekt yaratilganda avtomatik ravishda chaqiriladigan sinf metodining maxsus turi. Konstruktorlar, odatda, standart / foydalanuvchi tomonidan berilgan qiymatlarga ega bo‘lgan sinf a'zolarining o‘zgaruvchilarini initsializatsiyalash yoki ishlatilayotgan sinf uchun zarur bo‘lgan konfiguratsiya bosqichlarini bajarish uchun ishlatiladi (masalan, ma'lum bir fayl yoki ma'lumotlar bazasini ochish).

Oddiy metodlardan farqli o‘laroq, konstruktorlar ularni nomlashning ma'lum qoidalariga ega:

-Konstruktorlar har doim sinf bilan bir xil nomga ega bo‘lishi kerak (katta va kichik harflar hisobga olinadi);
-Konstruktorlar qaytish turiga ega emas (hatto void ham emas).

STANDART KONSTRUKTORLAR

Parametrlari bo‘lmagan konstruktor (yoki jimlik bo‘yicha parametrlarga ega) ***standart konstruktor*** deb ataladi. Agar foydalanuvchi tomonidan initsializatsiyalash uchun hech qanday qiymat ko‘rsatilmagan bo‘lsa, standart konstruktor chaqiriladi. Masalan:

```
#include <iostream>
class Fraction
{
private:
    int m_numerator;
    int m_denominator;

public:
    Fraction() // jimlik bo‘yicha konstruktor
    {
        m_numerator = 0;
        m_denominator = 1;
    }
```

```
    int getNumerator() { return m_numerator; }
    int getDenominator() { return m_denominator; }
    double getValue() { return (double)m_numerator / m_denominator; }
};


int main()
{
    Fraction drob; // argumentlar bo‘lmaganligi sababli, standart Fraction () konstruktori chaqiriladi
cout << drob.getNumerator() << "/" << drob.getDenominator() << '\n';

    return 0;

}
```

Bu sinf alohida int qiymatlari sifatida haqiqiy sonlarni o‘z ichiga oladi. Standart konstruktor Fraction deb nomlanadi (xuddi sinf kabi). Biz Fraction sinfining obyektini argumentlarsiz yaratganimiz uchun, standart konstruktor obyekt uchun xotira ajratilgandan so‘ng darhol ishladi va obyektimizni initsializatsiyalaydi.

PARAMETRLI KONSTRUKTORLAR

Standart konstruktor sinflarimizni standart qiymatlarga o‘tkazilishini ta'minlash uchun juda yaxshi bo‘lsa-da, sinfimiz obyektlari uchun ma'lum qiymatlarga ega bo‘lish kerak bo‘ladi, biz buni keyinroq beramiz. Yaxshiyamki, konstruktorlar parametrlar bilan ham e'lon qilinishi mumkin. Surat va maxrajni initsializatsiyalash uchun ishlatiladigan ikkita butun sonli konstruktor misoli Listingda berilgan.

```
#include <cassert>
using namespace std;
class Fraction
{
private:
```

```
    int m_numerator;
    int m_denominator;
public:
    Fraction()
    {
        m_numerator = 0;
        m_denominator = 1;
    }

    // Ikki parametrli konstruktor, ulardan biri standart qiymatga ega
    Fraction(int numerator, int denominator=1)
    {
        assert(denominator != 0);
        m_numerator = numerator;
        m_denominator = denominator;
    }

    int getNumerator() { return m_numerator; }
    int getDenominator() { return m_denominator; }
```

```
    double getValue() { return (double) m_numerator / m_denominator; }
};
```

Parametrli konstruktor foydalanish oson. Buning uchun to’g’ridan-to’g’ri initsializatsiyadan foydalanish kerak.

```
int a(7); // to‘g‘ridan-to‘g‘ri initsializatsiya
Fraction drob(4, 5); //to‘g‘ridan-to‘g‘ri initsializatsiya, Fraction (int, int) konstruktori chaqiriladi
```

Bu yerda kasrni 4 va 5 raqamlari bilan initsializatsiya qildik, natija 4/5.

C++11 standarti bo’yicha uniform-initsializatsiyadan foydalanishimiz mumkin:

```
int a { 7 }; // uniform-initsializatsiya
Fraction drob {4, 5};
```

Parametrli konstruktor uchun faqat bitta parametrni belgilashimiz mumkin, ikkinchi qiymat esa standart qiymat bo‘ladi:

Fraction seven(7);

Konstruktorlar uchun standart qiymatlar boshqa funksiyalar bilan bir xil ishlaydi, shuning uchun yuqoridagi misolda seven(7) ni chaqirganimizda, ikkinchi parametr 1 (standart) bo‘lgan Fraction (int, int) chaqiriladi.

OSHKORMAS YARATILGAN STANDART KONSTRUKTOR

Agar sinfingizda konstruktorlar bo‘lmasa, C++ avtomatik ravishda sinfingiz uchun umumiy standart konstruktorni yaratadi. Ba'zan uni yashirin konstruktor (yoki "oshkormas tarzda yaratilgan konstruktor") deb atashadi. Listingda berilgan sinfni ko‘rib chiqaylik.

```
class Date
{
private:
    int m_day = 12;
    int m_month = 1;
    int m_year = 2018;
};
```

Bu sinfda konstruktor yo‘q, shuning uchun kompilyator quyidagi konstruktorni yaratadi:

```
class Date
{
private:
    int m_day = 12;
    int m_month = 1;
    int m_year = 2018;

public:
    Date() {} };
```

Bu konstruktor sinf obyektlarini yaratishga imkon beradi, lekin ularni initsializatsiyalamaydi yoki sinf a'zolariga qiymatlar tayinlamaydi.

Aniq yaratilmagan konstruktorni ko‘ra olmasangiz ham, uning mavjudligini isbotlashingiz mumkin:

```
class Date
{
private:
    int m_day = 12;
    int m_month = 1;
    int m_year = 2018;

    // Hech qanday konstruktor ta'minlanmagan, shuning uchun C++ avtomatik ravishda umumiy standart konstruktorni yaratadi
};

int main()
{
    Date date; // yashirin konstruktorni chaqiriladi
    return 0; }
```

Yuqoridagi kod kompilyatsiya qilinadi, chunki yopiq konstruktor (umumiy bo‘lgan) Date obyektini yoqadi. Agar sinfingizda boshqa konstruktorlar bo‘lsa, u holda yashirin tarzda yaratilgan konstruktor yaratilmaydi(-listing).

```
class Date
{
private:
    int m_day = 12;
    int m_month = 1;
    int m_year = 2018;

public:
    Date(int day, int month, int year) // oddiy konstruktor (jimlik bo‘yicha emas)
    {
        m_day = day;
        m_month = month;
        m_year = year;
    }
```

```
    // Yashirin konstruktor yaratilmaydi, chunki allaqachon konstruktorimizni aniqlaganmiz
};

int main()
{
    Date date; // xato: obyektni yaratib bo‘lmaydi, chunki standart konstruktor yo‘q va kompilyator avtomatik ravishda konstruktorni yaratmagan.
    Date today(14, 10, 2020); // today obyektini initsializatsiyalash
    return 0;
}
```

Har bir sinf uchun har doim kamida bitta konstruktor yaratish tavsiya etiladi. Bu sizga o‘z sinfingiz obyektlarini yaratish jarayonini boshqarishga imkon beradi va boshqa konstruktorlarni qo‘shgandan so‘ng yuzaga kelishi mumkin bo‘lgan muammolarni oldini oladi.

SINF TARKIBIDAGI A’ZO O’ZGARUVCHILARNI INITSIALIZATSIYALASH

Quyida C++ da initsializatsiya ro‘yxati yordamida sinf a'zolarining o‘zgaruvchilarini qanday initsializatsiyalashni, shuningdek, bu holda yuzaga kelishi mumkin bo‘lgan xususiyatlar va nyuanslarni ko‘rib chiqamiz.

Yuqorida sinfimiz a'zolarini konstruktorda ta’minlash operatori orqali initsializatsiya qildik:

```
class Values
{
private:
    int m_value1;
    double m_value2;
    char m_value3;

public:
    Values()
    {
```

```
        // Bularning hammasi initsializatsiya emas, balki ta’minlash amallari
        m_value1 = 3;
        m_value2 = 4.5;
        m_value3 = 'd';
    }
};
```

Birinchidan, m_value1, m_value2 va m_value3 yaratiladi. Keyin konstruktor tanasi bajariladi, bu yerda bu o‘zgaruvchilarga qiymatlar beriladi. Kod obyektga yo‘naltirilmagan C++ ga o‘xshash (-listing).

```
int m_value1;
double m_value2;
char m_value3;

m_value1 = 3;
m_value2 = 4.5;
m_value3 = 'd';
```

C++ tilining sintaksisi nuqtai nazaridan, hech qanday savol yo‘q - hamma narsa to‘g‘ri, lekin e’lon qilishdan keyin o‘zlashtirishni emas, balki initsializatsiyani ishlatish samaraliroq.

Oldingi mavzularda bilganimizdek, ba'zi ma'lumotlar turlarini (masalan, konstantalarni) zudlik bilan initsializatsiyalash kerak. Quyidagi misolni ko‘rib chiqaylik:

```
class Values
{
private:
    const int m_value;

public:
    Values()
    {
        m_value = 3; // xato: konstantalar qiymatlar tayinlanishi mumkin emas
    }
};
```

Kod obyektga yo‘naltirilmagan C++ga o‘xshash:

```
const int m_value; // xato: konstantalar qiymatlar bilan boshlanishi kerak
m_value = 7; // xato: konstantalarga qiymatlar tayinlanishi mumkin emas
```

Ushbu muammoni hal qilish uchun, C++ o’zgaruvchilar e’lonidan keyin ularga qiymatlar berish o‘rniga**, a'zolarni initsializatsiyalash ro‘yxati** orqali sinf a'zolarining o‘zgaruvchilarini initsializatsiyalash metodini qo‘shdi. Bu ro‘yxatni **massivlarni** initsializatsiyalash uchun ishlatiladigan shunga o‘xshash boshlovchi ro‘yxati bilan adashtirmaslik lozim. Quyidagi uch xil usul mavjud:

```
int value1 = 3; // initsializatsiyani nusxalash
double value2(4.5); // to‘g‘ridan-to‘g‘ri initsializatsiya
char value3 {'d'} // uniform-initsializatsiya
```

Initsializatsiya ro‘yxatini ishlatish to‘g‘ridan-to‘g‘ri initsializatsiya bilan deyarli bir xil (yoki C++ 11 da uniform-initsializatsiya).

Buni aniqroq qilish uchun bir misolni ko‘rib chiqaylik. Bu yerda (-listingda) konstruktordagi sinf a'zolari o‘zgaruvchilariga qiymatlar belgilash kodi berilgan.

```
class Values
{
private:
    int m_value1;
    double m_value2;
    char m_value3;

public:
    Values()
    {
        // Bu ta’minlash operatori, initsializatsiya emas
        m_value1 = 3;
        m_value2 = 4.5;
        m_value3 = 'd';
    }
};
```

Keling, ushbu kodni qayta yozamiz, lekin bu safar initsializatsiya ro‘yxati yordamida:

```
#include <iostream>
using namespace std
class Values
{
private:
    int m_value1;
    double m_value2;
    char m_value3;

public:
    Values() : m_value1(3), m_value2(4.5), m_value3('d') // biz to‘g‘ridan-to‘g‘ri sinfning a'zo o‘zgaruvchilarini initsializatsiya qilamiz
    {
    // Ta’minlash operatorini ishlatishning hojati yo‘q
    }

    void print()
```

```
        {
            cout << "Values(" << m_value1 << ", " << m_value2 << ", " <<
m_value3 << ")\n";
        }
    };

    int main()
    {
        Values value;
        value.print();
        return 0;
    }
```

Initsializatsiya ro‘yxati konstruktor parametrlaridan so‘ng darhol yoziladi. U ikki nuqta (:) bilan boshlanadi , so‘ngra har bir o‘zgaruvchining qiymati qavs ichida ko‘rsatiladi. Endi konstruktor tanasida ta’minlash operatsiyalarini bajarish shart emas. Shuni ham unutmangki, a'zolarni initsializatsiyalash ro‘yxati nuqta-vergul bilan tugamaydi.

Shuningdek, obyekt yaratilganda initsializatsiyalash uchun qiymatlarni uzatish qobiliyatini qo‘shishingiz ham mumkin:

Values value(3, 4.5); // value1 = 3, value2 = 4.5, value3 = 'd' (jimlik bo’yicha qiymatlar)

Agar foydalanuvchi ularni bermagan bo‘lsa, biz **standart parametrlardan** foydalanishimiz ham mumkin. Masalan, konstanta a'zo o‘zgaruvchiga ega bo‘lgan sinf:

```
class Values
{
private:
    const int m_value;

public:
    Values(): m_value(7) // konstanta a'zo o‘zgaruvchini to‘g‘ridan-to‘g‘ri initsializatsiya qiladi
    {
    } };
```

Bu ishlaydi, chunki bizga konstanta o‘zgaruvchilarni ishga tushirishga ruxsat berilgan (lekin e’lon qilishdan keyin ularga qiymatlar tayinlanmaydi).

C++11 da uniform-initsializatsiya. C ++ 11 da to‘g‘ridan-to‘g‘ri initsializatsiya o‘rniga uniform-initsializatsiyadan ham foydalanish mumkin:

```
class Values
{
private:
    const int m_value;

public:
    Values(): m_value { 7 } // uniform-initsializatsiya
    {
    }
};
```

Ushbu sintaksisdan foydalanish tavsiya etiladi (agar sinfingizning a'zolarining o‘zgaruvchilari sifatida doimiy yoki mos yozuvlar ishlatmasangiz ham), chunki a'zolarni initsializatsiyalash ro‘yxatlari

kompozitsiya va meros uchun zarur (Buni keyingi mavzularda ko‘rib chiqamiz).

Sinfdagi massivlarni initsializatsiyalash. Bir o‘lchamli massivli sinfni a'zo o‘zgaruvchi sifatida ko‘rib chiqaylik:

```
class Values
{
private:
    const int m_array[7];
};
```

C++11dan oldin, biz faqat initsializatsiyalash ro‘yxati orqali massivni nol qila olamiz:

```
class Values
{
private:
    const int m_array[7];

public:
    Values(): m_array {} // massivga nolni joylashtirish
```

```
    {
    // Agar biz massiv qiymatlarga ega bo‘lishini xohlasak, bu yerda ta’minlashdan foydalanishimiz kerak
    }
};
```

Biroq, C++ 11 da siz uniform-initsializatsiya yordamida massivni to‘liq initsializatsiyalashingiz ham mumkin:

```
class Values
{
private:
    const int m_array[7];

public:
    Values(): m_array { 3, 4, 5, 6, 7, 8, 9 } // massivni initsializatsiyalash uchun uniform-initsializatsiyadan foydalanish
    {

    }
};
```