

## Конструктори

Коли всі члени класу (або структури) є відкритими, то ми можемо ініціювати клас (або структуру) безпосередньо, використовуючи список ініціалізаторів.

```
1 class Boo
2 {
3 public:
4     int m_a;
5     int m_b;
6 };
7
8 int main()
9 {
10     Boo boo1 = { 7, 8 }; // список инициализаторов
11     Boo boo2 { 9, 10 }; // uniform инициализация (C++11)
12
13     return 0;
14 }
```

Однак, як тільки ми зробимо будь-які змінні-члени класу закритими, то ми більше не зможемо ініціювати їх безпосередньо

**Конструктор** - це особливий тип методу класу, який автоматично викликається при створенні об'єкта цього ж класу. Конструктори зазвичай використовуються для ініціалізації змінних-членів класу значеннями, які надані за замовчуванням / користувачем, або для виконання будь-яких кроків налаштування, необхідних для використовуваного класу (наприклад, відкрити певний файл або базу даних).

На відміну від звичайних методів, конструктори мають певні правила щодо їх імен :

- конструктори завжди повинні мати те ж ім'я, що і клас (враховуються верхній і нижній регістри).
- Конструктори не мають типу повернення (навіть void).

Конструктор, який не має параметрів (або має параметри, всі з яких мають значення за замовчуванням), називається **конструктором за замовчуванням**. Він викликається, якщо користувачем не вказані значення для ініціалізації. наприклад:

```

1 #include <iostream>
2
3 class Fraction
4 {
5 private:
6     int m_numerator;
7     int m_denominator;
8
9 public:
10    Fraction() // конструктор по умолчанию
11    {
12        m_numerator = 0;
13        m_denominator = 1;
14    }
15
16    int getNumerator() { return m_numerator; }
17    int getDenominator() { return m_denominator; }
18    double getValue() { return static_cast<double>(m_numerator) / m_denominator; }
19 };
20
21 int main()
22 {
23     Fraction drob; // так как нет никаких аргументов, то вызывается конструктор по умолчанию Fraction()
24     std::cout << drob.getNumerator() << "/" << drob.getDenominator() << '\n';
25
26     return 0;
27 }

```

Цей клас містить дріб у вигляді окремих значень типу `int`. Конструктор за замовчуванням називається `Fraction` (як і клас). Оскільки ми створили об'єкт класу `Fraction` без аргументів, то конструктор за замовчуванням спрацював відразу ж після виділення пам'яті для об'єкта, і ініціалізувати наш об'єкт.

Результат виконання програми вище:

0 / 1

Чисельник (`m_numerator`) і знаменник (`m_denominator`) були ініційовані значеннями, які поставлені в конструкторі за замовчуванням! Це настільки корисно, що майже кожен клас має свій конструктор за замовчуванням. Без нього значеннями чисельника і знаменника був би сміття до тих пір, поки ми б явно не присвоїли їм нормальні значення.

## Конструктори з параметрами

Хоча конструктор за замовчуванням відмінно підходить для забезпечення ініціалізації класів значеннями за замовчуванням, часто може бути потрібно, щоб екземпляри класу мали певні значення, які надаються пізніше. На щастя, конструктори також можуть бути оголошені з параметрами. Ось приклад конструктора, який має два цілочисельних параметра, які використовуються для ініціалізації чисельника і знаменника:

```

1 #include <cassert>
2
3 class Fraction
4 {
5     private:
6         int m_numerator;
7         int m_denominator;
8
9     public:
10        Fraction() // конструктор по умолчанию
11        {
12            m_numerator = 0;
13            m_denominator = 1;
14        }
15
16        // Конструктор с двумя параметрами, один параметр имеет значение по умолчанию
17        Fraction(int numerator, int denominator=1)
18        {
19            assert(denominator != 0);
20            m_numerator = numerator;
21            m_denominator = denominator;
22        }
23
24        int getNumerator() { return m_numerator; }
25        int getDenominator() { return m_denominator; }
26        double getValue() { return static_cast<double>(m_numerator) / m_denominator; }
27 };

```

Зверніть увагу, тепер у нас є два конструктора: конструктор за замовчуванням, який буде викликатися, якщо ми не надамо значення, і конструктор з параметрами, який буде викликатися, якщо ми надамо значення. Ці два конструктора можуть мирно існувати в одному класі завдяки перевантаженню функцій. Фактично, ви можете визначити будь-яку кількість конструкторів до тих пір, поки у них будуть унікальні параметри (враховується їх кількість і тип).

Як використовувати конструктор з параметрами? Все просто! Пряма ініціалізація:

```

1 int a(7); // инициализируем напрямую
2 Fraction drob(4, 5); // инициализируем напрямую, вызывается конструктор Fraction(int, int)

```

У C++ 11 ми також можемо використовувати uniform ініціалізацію:

```

1 int a { 7 }; // uniform инициализация
2 Fraction drob {4, 5}; // uniform инициализация, вызывается конструктор Fraction(int, int)

```

## Зменшення кількості конструкторів

В наведеному вище прикладі з класом Fraction і двома конструкторами (за замовчуванням і з параметрами), конструктор за замовчуванням, насправді, зайвий. Ми могли б спростити цей клас наступним чином:

```

1  #include <cassert>
2
3  class Fraction
4  {
5  private:
6      int m_numerator;
7      int m_denominator;
8
9  public:
10     // Конструктор по умолчанию
11     Fraction(int numerator=0, int denominator=1)
12     {
13         assert(denominator != 0);
14         m_numerator = numerator;
15         m_denominator = denominator;
16     }
17
18     int getNumerator() { return m_numerator; }
19     int getDenominator() { return m_denominator; }
20     double getValue() { return static_cast<double>(m_numerator) / m_denominator; }
21 };

```

Хоча цей конструктор і раніше є конструктором за замовчуванням, він тепер визначений таким чином, що може приймати одне або два значення, надані користувачем:

```

1  Fraction drob; // ВИЗОВ Fraction(0, 1)
2  Fraction seven(7); // ВИЗОВ Fraction(7, 1)
3  Fraction sixTwo(6, 2); // ВИЗОВ Fraction(6, 2)

```

## Неявно генерований конструктор за замовчуванням

Якщо ваш клас не має інших конструкторів, то C++ автоматично згенерує для вашого класу відкритий конструктор за замовчуванням. Його іноді називають неявним конструктором (або ще «неявно згенерував конструктором»). Розглянемо наступний клас:

```

1  class Date
2  {
3  private:
4      int m_day = 12;
5      int m_month = 1;
6      int m_year = 2018;
7  };

```

У цього класу немає конструктора. Тому компілятор згенерує наступний, ідентичний з виконання, конструктор:

```

1 class Date
2 {
3 private:
4     int m_day = 12;
5     int m_month = 1;
6     int m_year = 2018;
7
8 public:
9     Date() // неявно генерируемый конструктор
10    {
11    };
12 };

```

Цей конструктор дозволяє створювати об'єкти класу, але не виконує їх ініціалізацію і не присвоює значення членам класу.

Хоча ви не можете побачити неявно згенерований конструктор, але його існування можна довести:

```

1 class Date
2 {
3 private:
4     int m_day = 12;
5     int m_month = 1;
6     int m_year = 2018;
7
8     // Не было предоставлено конструктора, поэтому C++ автоматически создаст открытый конструктор по умолчанию
9 };
10
11 int main()
12 {
13     Date date; // вызов неявного конструктора
14
15     return 0;
16 }

```

Код вище скомпілюється, оскільки в об'єкті date спрацює неявний конструктор (який є відкритим). Якщо ваш клас має інші конструктори, то неявно генерований конструктор створюватися не буде, наприклад:

```

1 class Date
2 {
3 private:
4     int m_day = 12;
5     int m_month = 1;
6     int m_year = 2018;
7
8 public:
9     Date(int day, int month, int year) // обычный конструктор (не по умолчанию)
10    {
11        m_day = day;
12        m_month = month;
13        m_year = year;
14    }
15
16    // Неявный конструктор не создается, так как мы уже определили свой конструктор
17 };
18
19 int main()
20 {
21     Date date; // ошибка: невозможно создать объект, так как конструктор по умолчанию не существует
22     Date today(14, 10, 2020); // инициализируем объект today
23
24     return 0;
25 }

```

Рекомендується завжди створювати принаймні один конструктор в класі. Це дозволить вам контролювати процес створення об'єктів вашого класу, і запобігатиме виникненню потенційних проблем після додавання інших конструкторів.

### Завдання №3

*Напишіть клас Ball, який повинен мати такі дві закриті змінні-члени зі значеннями за замовчуванням:*

*m\_color( Red);  
m\_radius( 20.0).*

*У Ball-е повинні бути наступні конструктори:*

*для встановлення значення тільки для m\_color;*

*для встановлення значення тільки для m\_radius;*

*для встановлення значень і для m\_radius, і для m\_color;*

*для встановлення значень, коли значення не надані взагалі.*

Наступний код функції main ():

```
1 int main()
2 {
3     Ball def;
4     def.print();
5
6     Ball black("black");
7     black.print();
8
9     Ball thirty(30.0);
10    thirty.print();
11
12    Ball blackThirty("black", 30.0);
13    blackThirty.print();
14
15    return 0;
16 }
```

Програма повинна видавати такий результат:

```
color: red, radius: 20
color: black, radius: 20
color: red, radius: 30
color: black, radius: 30
```