Задача "Обобщенный оператор сравнения"

Екатеринбург 2023

Обобщенный оператор сравнения

Обе задачи необходимо решать с использованием компилятора C++17 и выше.

Условие задачи невероятно просто. Вам дан класс, который содержит два оператора сравнения с любым произвольным типом:

```
class NewA
{
public:
    NewA(int a, int b) : m_a(a), m_b(b){}

    // Операторы сравнения с NewA
    bool operator<(const NewA& other) const
    {
        return (m_a < other.m_a) && (m_b < other.m_b);
    }

    bool operator>(const NewA& other) const
    {
        return (m_a > other.m_a) && (m_b > other.m_b);
    }

    // Операторы сравнения с int
    bool operator<(const int other) const
    {
        return m_a < other;
    }

    bool operator>(const int other) const
    {
        return m_a > other;
    }

private:
    int m_a = 0;
    int m_b = 0;
};
```

Необходимо написать шаблонный класс, который будет дополнять операторы сравнения до полного набора: т.е. в данном примере, реализованы операторы $<$ и $>$ для типов `NewA` и `int`. Операторы, которые должны быть реализованы дополнительно: $<=$, $>=$, $==$, $!=$.

Стоит отметить, что исходный класс изменять нельзя, кроме внесения операторов $<$ и $>$ для других типов. Более того, исходный класс может быть дополнен другими операторами $<$ и $>$ для других типов(на данный момент операторы $<$ и $>$ определены для типов `NewA` и `int`, но могут быть и другие типы, например, `char`).

```
class NewA : public OtherComparisionOperators<NewA>
{
public:
    NewA(int a, int b) : m_a(a), m_b(b){}

    // Операторы сравнения с NewA
    bool operator<(const NewA& other) const
    {
        return (m_a < other.m_a) && (m_b < other.m_b);
    }

    bool operator>(const NewA& other) const
    {
        return (m_a > other.m_a) && (m_b > other.m_b);
    }

    // Операторы сравнения с int
    bool operator<(const int other) const
    {
        return m_a < other;
    }

    bool operator>(const int other) const
    {
        return m_a > other;
    }

private:
    int m_a = 0;
    int m_b = 0;
```

```
};

int main()
{
    NewA a1(1, 2);
    NewA a2(2, 3);
    std::cout << (a1 >= a2) << (a1 <= a2) << (a1 == a2) << (a1 != a2);
    std::cout << (a1 >= 4) << (a1 <= 4) << (a1 == 4) << (a1 != 4);

    //Перегрузите < и > для double
    //std::cout << (a1 >= 4.0) << (a1 <= 4.0) << (a1 == 4.0) << (a1 != 4.0);
}
```

Еще одно важное уточнение, вместо класса `NewA` может быть любой другой класс, например:

```
class NewB : public OtherComparisionOperators<NewB>
{
public:
    NewB(std::string_view val) : m_stringView(val){}

    // Операторы сравнения с std::string
    bool operator<(const std::string& other) const
    {
        return m_stringView < other;
    }

    bool operator>(const std::string& other) const
    {
        return m_stringView > other;
    }

private:
    std::string_view m_stringView;
};
```

Для успешной сдачи данной задачи:

1. Реализовать класс `OtherComparisionOperators`;
2. Написать тесты для вашего функционала

Подсказка: для реализации данной задачи, вам поможет идиома языка C++, которая называется CRTP.

Список литературы

[1] Вандервуд Д., Джосаттис Н., Грегор Д. *Шаблоны C++. Справочник разработчика 2-е изд.* - СПБ.: ООО «Диалектика», 2020

[2] Пикус Ф.Г. *Идиомы и паттерны проектирования в современном C++* - М.: ДМК Пресс, 2020

[3] Старая новая техника: CRTP `http://scrutator.me/post/2014/06/26/crtp_demystified.aspx`

Задача под звездочкой. И немного мета магии...

Данная задача имеет исследовательский характер. Вам дан следующий код:

```
template <typename T, typename ...OtherTypes>
struct any_of
{
    static constexpr bool value = (std::is_same_v<T, OtherTypes> || ...);
};

template <typename T, typename ...OtherTypes>
constexpr bool any_of_v = any_of<T, OtherTypes...>::value;

struct A : OtherComparisionOperators<A>
{
public:
    A(int a, int b) : m_a(a), m_b(b){}

    template <typename OtherT>
    std::enable_if_t<any_of_v<OtherT, A, B>, bool>
    operator<(const OtherT& other) const
    {
        return (m_a < other.m_a) && (m_b < other.m_b);
    }

    template <typename OtherT>
    std::enable_if_t<any_of_v<OtherT, A, B>, bool>
    operator>(const A& other) const
    {
        return (m_a > other.m_a) && (m_b > other.m_b);
    }

    template <typename OtherT>
    std::enable_if_t<std::is_integral_v<OtherT>, bool>
    operator<(const OtherT& other) const
    {
        return m_a < other;
    }
```

```
35
36      template <typename OtherT>
37      std::enable_if_t<std::is_integral_v<OtherT>, bool>
38      operator>(const OtherT& other) const
39      {
40          return m_a > other;
41      }
42
43
44  private:
45      int m_a = 0;
46      int m_b = 0;
47  };
```

Необходимо:

1. Понять с какими типами может быть сравним клас `A`. Реализовать тип `B`, с которым может быть сравним класс `A`.
2. Дополнить реализацию класса `OtherComparisionOperators`, чтобы он мог работать с классом A

Какие темы необходимо рассмотреть, чтобы решить данную задачу:

1. SFINAE
2. Вариативные шаблоны
3. Fold expression

Список литературы

[1] Вандервуд Д., Джосаттис Н., Грегор Д. *Шаблоны C++. Справочник разработчика 2-е изд.* - СПБ.: ООО «Диалектика», 2020

[2] Пикус Ф.Г. *Идиомы и паттерны проектирования в современном C++* - М.: ДМК Пресс, 2020

[3] SFINAE. Как много в этом слове: `http://scrutator.me/post/2016/12/12/sfinae.aspx`