Quiz: semester 2, week 4

Студент _____ *группы* _____

Я подтверждаю что ответы ниже являются моими собственными и даны честно _____

(число, подпись)

Пожалуйста не используйте бланк теста для ответов, возьмите отдельный лист.
Внимательно изучите следующий код, вопросы ниже касаются в основном его особенностей.

```
#include <cassert>

template <typename T, typename U> bool
operator == (const T &x, const U &y)
{
  return !(x < y) && !(y < x);
}

template <typename T, typename U> bool
operator != (const T &x, const U &y)
{
  return !(x == y);
}

struct Pair
{
  int m_x, m_y;
  Pair (int x, int y = 0): m_x(x), m_y(y) {}
};

bool
operator < (const Pair &lhs, const Pair &rhs)
{
  return (lhs.m_x < rhs.m_x) || ((lhs.m_x == rhs.m_x) && (lhs.m_y < rhs.m_y));
}

int
main (void)
{
  Pair a (2);
  Pair b (2, 1);
  int c = 2, d = 2;
  /* 1 */ assert (b != a);
  /* 2 */ assert (c == d);
  /* 3 */ assert (a == c);
  /* 4 */ assert (d == b);
  return 0;
}
```

Везде ниже предполагается, что все asserts включены.

Q1: Охарактеризуйте как будет скомпилирован и исполнен этот код:

1) скомпилирован и исполнен без проблем
2) ошибка компиляции
3) assertion failure
4) run-time error

Q2 (PA): Напишите реализацию обобщенного operator > так, чтобы прошли следующие проверки:

```
assert (b > a);
assert (!(c > a));
assert (a > Pair (1, 3));
```

Q3: Есть ли среди условий 1-4 (помечены комментариями) принципиально проваливающие assertion (даже если до его проверки в реальном исполнении не дойдет)? Пометьте все (может быть больше одного или ни одного ответа).
1) 1
2) 2
3) 3
4) 4

Q4: Сколько раз (до завершения программы или первой ошибки) вызовется шаблонная функция `operator ==` ?
1) ни разу, программа не скомпилируется
2) дважды
3) трижды
4) четырежды
5) более четырех раз

Q5: Студент Шустрый решил внести оператор < внутрь класса Pair следующим образом:

```
struct Pair
{
  int m_x, m_y;
  Pair (int x, int y = 0): m_x(x), m_y(y) {}
  bool operator < (const Pair &rhs)
  {
    return (m_x < rhs.m_x) || ((m_x == rhs.m_x) && (m_y < rhs.m_y));
  }
};
```

Это приведет к:
1) ошибке компиляции
2) появлению новых assertion failures
3) run-time error
4) ничего не изменится.

Q6: Отказавшись от прошлой идеи, студент Шустрый решил вместо этого сделать конструктор Pair explicit:

```
struct Pair
{
  int m_x, m_y;
  explicit Pair (int x, int y = 0): m_x(x), m_y(y) {}
};
```

Это приведет к ошибке компиляции из-за того, что:
1) explicit в этом констексте является ошибкой
2) explicit конструктор не может быть пустым
3) внутри условий assertions используется неявное приведение типов
4) ошибки компиляции не произойдет

Q7 (PA): Напишите определение структуры Triple и оператора < для неё, чтобы (при неизменных обобщенных операторах сравнения) прошли следующие проверки:

```
Triple e (2, 0, 0);
Triple f (2, 0, 0);
assert (e == f);
assert (2 == e);
assert (Triple(2, 1, 0) != e);
assert (Triple(2, 0, 1) != f);
```

Q8: Профессор Вальяжный предлагает заменить явный шаблонный полиморфизм на CRTP:

```
template <typename T> struct OperatorMixin {};

template <typename T> bool
operator == (const OperatorMixin<T>& lhs, const OperatorMixin<T>& rhs)
{
    const T& d1 = static_cast<const T&> (lhs);
    const T& d2 = static_cast<const T&> (rhs);

    return !(d1 < d2) && !(d2 < d1);
}

template <typename T> bool
operator != (const OperatorMixin<T>& o1, const OperatorMixin<T>& o2)
{
    return !(o1 == o2);
}

struct Pair : public OperatorMixin<Pair>
{
  int m_x, m_y;
  Pair (int x, int y = 0): m_x(x), m_y(y) {}
};

bool
operator < (const Pair &lhs, const Pair &rhs)
{
  return (lhs.m_x < rhs.m_x) || ((lhs.m_x == rhs.m_x) && (lhs.m_y < rhs.m_y));
}
```

К сожалению, чтобы сохранить консистентность с оригиналом это приведет к необходимости:

1) инвертировать некоторые assertion checks
2) добавить в некоторые assertion checks явное приведение типов
3) добавить в некоторые assertion checks явное конструирование объектов
4) это не приведет к необходимости менять assertion checks

Q9 (PA): Добавьте к коду профессора Вальяжного обобщенный оператор >= так чтобы прошли проверки

```
assert (b >= a);
assert (Pair(c) >= a);
```

Q10: Вернитесь к вопросу Q5. Теперь, если поверх CRTP из прошлого вопроса, студент Шустрый все-таки внесет метод operator< внутрь класса это приведет к:

1) ошибке компиляции
2) появлению новых assertion failures
3) run-time error
4) ничего не изменится

Q11 (ungraded): Какие ощущения оставил у вас этот тест
1) Глаза сейчас вытекут кровью и даже этот пункт я заполняю вслепую
2) Слишком легкий. Скажем, код из Q8 я мог бы написать сам. Кстати, хочу обратить ваше внимание, что `int main (void)` это форма main, которая допустима в стандарте C, но не в стандарте C++, так что у вас ошибка в формулировке задания
3) Я почему-то сейчас понял, что если пять лет писать компиляторы, то сходишь с ума
4) Ничего из вышеперечисленного, просто хочу узнать результаты