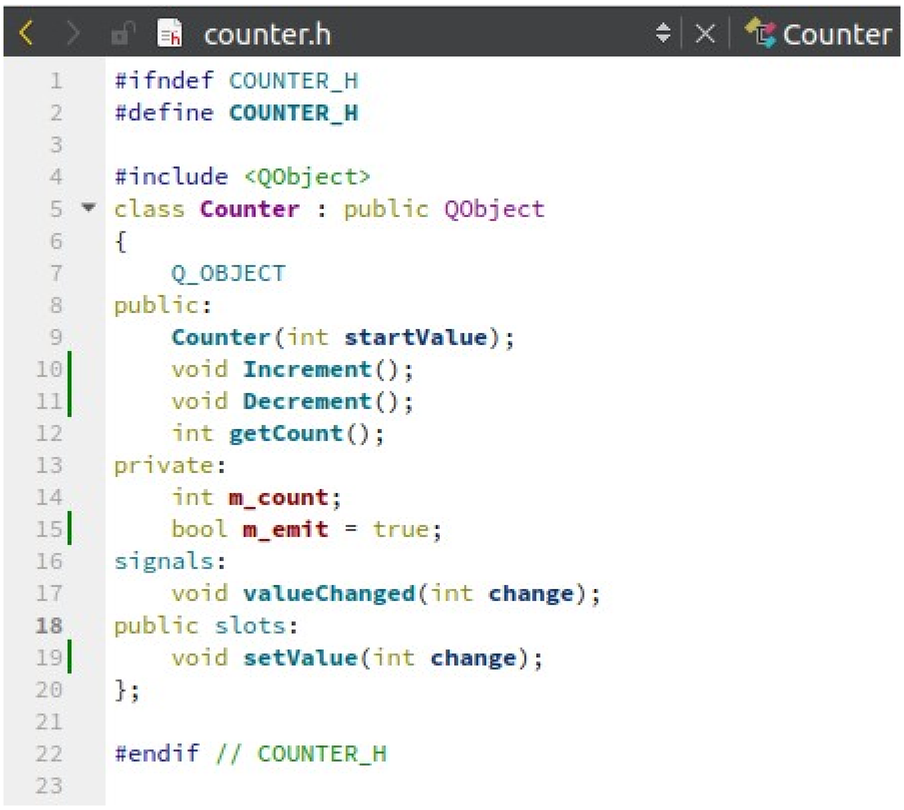
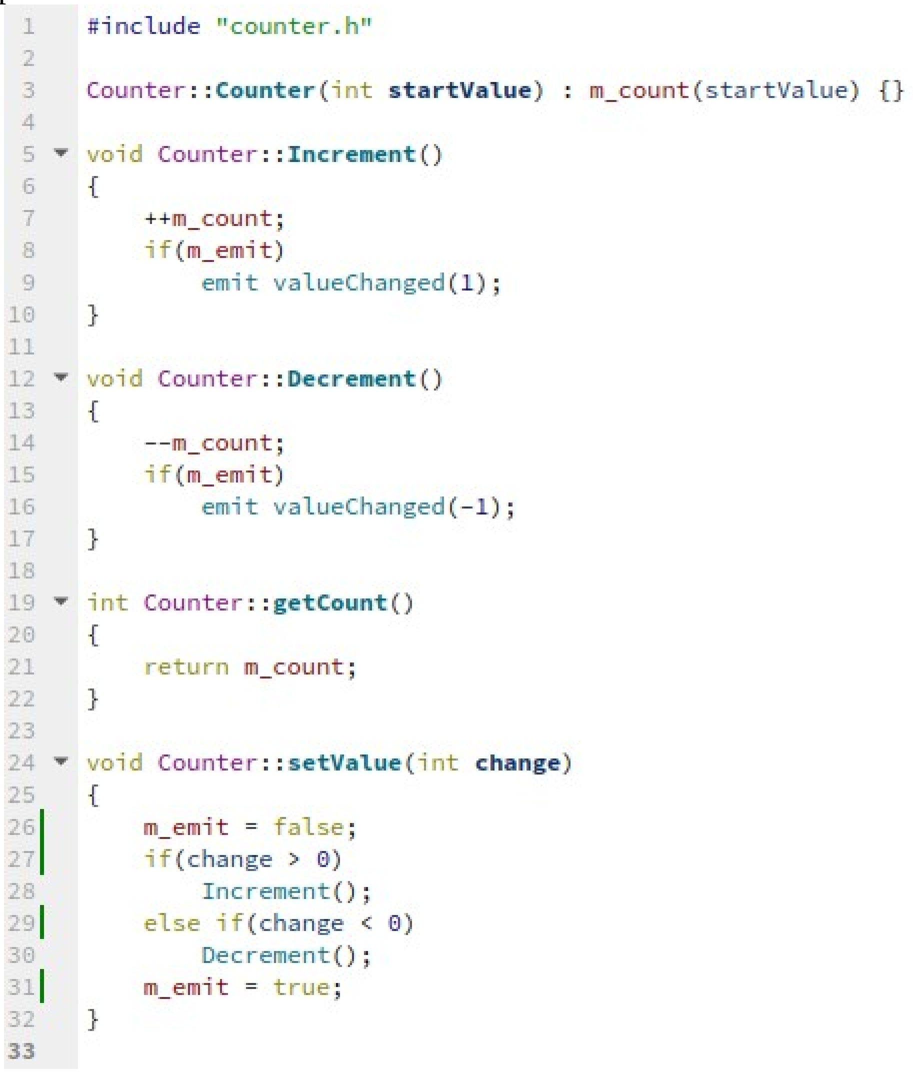
1. Counter.h Interface
It has mandatory interface:
- void Increment();
- void Decrement();
- Counter(int startValue);



```cpp
#ifndef COUNTER_H
#define COUNTER_H

#include <QObject>
class Counter : public QObject
{
    Q_OBJECT
public:
    Counter(int startValue);
    void Increment();
    void Decrement();
    int getCount();
private:
    int m_count;
    bool m_emit = true;
signals:
    void valueChanged(int change);
public slots:
    void setValue(int change);
};

#endif // COUNTER_H

```

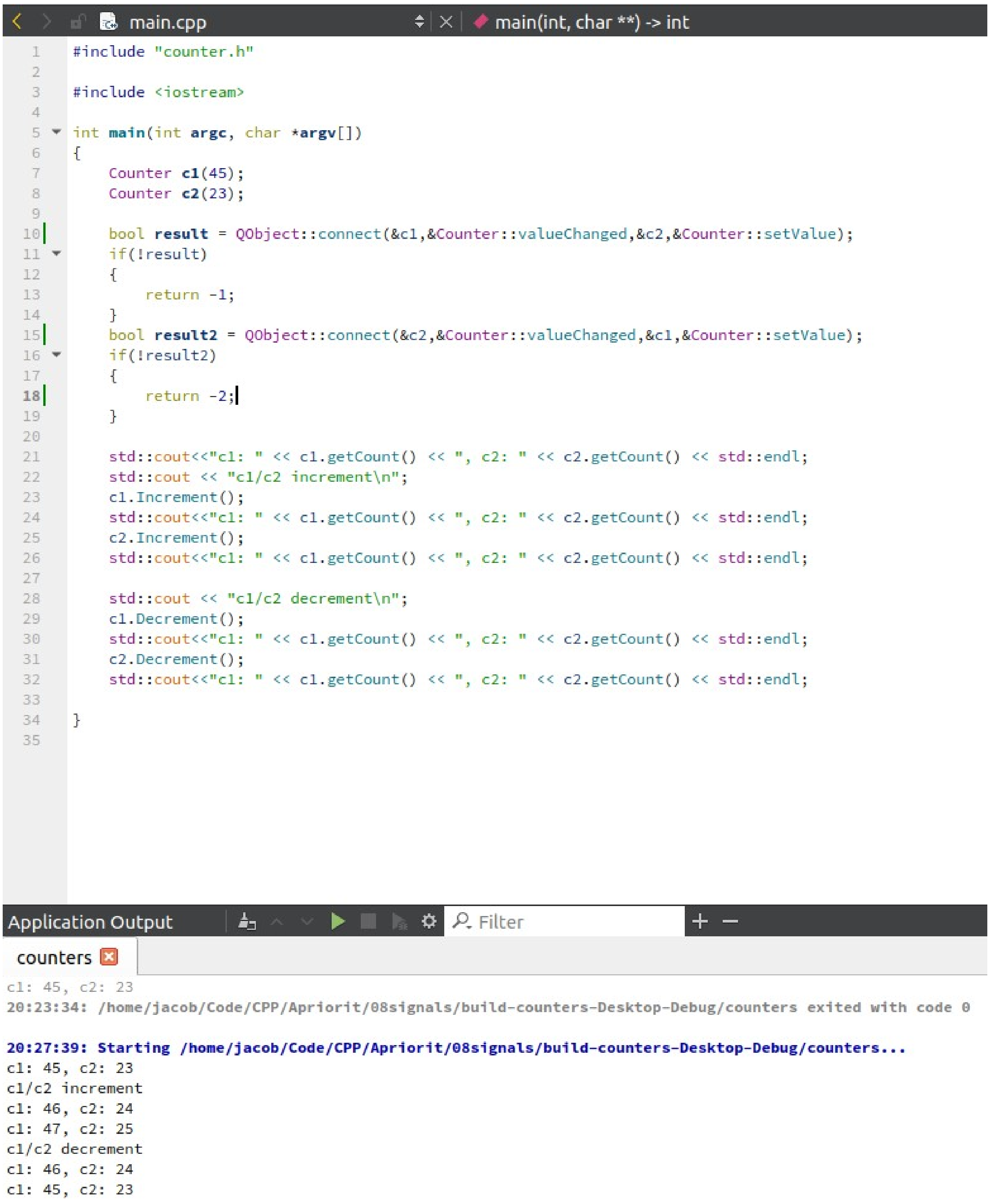
2. Counter.cpp



```cpp
#include "counter.h"

Counter::Counter(int startValue) : m_count(startValue) {}

void Counter::Increment()
{
    ++m_count;
    if(m_emit)
        emit valueChanged(1);
}

void Counter::Decrement()
{
    --m_count;
    if(m_emit)
        emit valueChanged(-1);
}

int Counter::getCount()
{
    return m_count;
}

void Counter::setValue(int change)
{
    m_emit = false;
    if(change > 0)
        Increment();
    else if(change < 0)
        Decrement();
    m_emit = true;
}

```

To prevent infinite loop I added m_emit boolean to guard whenever Increment/Decrement is called from slot or by user.
I could've typed ++m_count and –m_count in slot, but I chose to call Increment and Decrement methods. The reasoning for that is because the first option it is duplication of code (also breaking SOLID) and if this operation consisted from more than 1 line there would be even more code duplication.

3. Results



```cpp
#include "counter.h"

#include <iostream>

int main(int argc, char *argv[])
{
    Counter c1(45);
    Counter c2(23);

    bool result = QObject::connect(&c1,&Counter::valueChanged,&c2,&Counter::setValue);
    if(!result)
    {
        return -1;
    }
    bool result2 = QObject::connect(&c2,&Counter::valueChanged,&c1,&Counter::setValue);
    if(!result2)
    {
        return -2;
    }

    std::cout<<"c1: " << c1.getCount() << ", c2: " << c2.getCount() << std::endl;
    std::cout << "c1/c2 increment\n";
    c1.Increment();
    std::cout<<"c1: " << c1.getCount() << ", c2: " << c2.getCount() << std::endl;
    c2.Increment();
    std::cout<<"c1: " << c1.getCount() << ", c2: " << c2.getCount() << std::endl;

    std::cout << "c1/c2 decrement\n";
    c1.Decrement();
    std::cout<<"c1: " << c1.getCount() << ", c2: " << c2.getCount() << std::endl;
    c2.Decrement();
    std::cout<<"c1: " << c1.getCount() << ", c2: " << c2.getCount() << std::endl;

}
```

```
c1: 45, c2: 23
20:23:34: /home/jacob/Code/CPP/Apriorit/08signals/build-counters-Desktop-Debug/counters exited with code 0

20:27:39: Starting /home/jacob/Code/CPP/Apriorit/08signals/build-counters-Desktop-Debug/counters...
c1: 45, c2: 23
c1/c2 increment
c1: 46, c2: 24
c1: 47, c2: 25
c1/c2 decrement
c1: 46, c2: 24
c1: 45, c2: 23
```

The other solution I considered was using Qt::SingleShotConnection flag while creating connection, which would make creation of m_emit flag unnecessary. On the other hand it would require to connect counters each time before sending signals.