Использование pthread API

Ваша задача - реализовать классический паттерн producer-consumer с небольшими дополнительными условиями. Программа должна состоять из четырех потоков: один - главный, producer, consumer и interruptor. На стандартный ввод программе подается строка - список чисел произвольной длины.

- Задача producer-потока - получить на вход список чисел, и по очереди использовать каждое значение из этого списка для обновления переменной разделяемой между потоками. После этого поток должен дождаться реакции consumer-потока, и продолжить обновление значений только после того как поток-consumer принял это изменение. Функция исполняющая код этого потока producer_routine должна принимать указатель на объект типа Value, и использовать его для обновления.
- Задача consumer-потока отреагировать на каждое изменение переменной data и набирать сумму полученных значений. После того как достигнуто последнее обновление, функция потока должна вернуть результирующую сумму. Также этот поток должен защититься от попыток потока-interruptor его остановить. Функция исполняющая код этого потока consumer_routine должна принимать указатель на тот же объект типа Value, и читать из него обновления .
- Задача потока-interruptor проста: пока происходит процесс обновления значений, он должен постоянно пытаться остановить поток consumer. Как только поток producer произвел последнее обновление, этот поток завершается.

Функция run_threads должна запускать все три потока, дожидаться их выполнения, и возвращать результат потока-consumer.

Для обновления и получения значения следует использовать подготовленный класс Value. Следует создать один экземпляр Value, передать его аргументом в функции producer_routine и consumer_routine. Чтобы обновить значение, следует использовать метод update, чтобы получить - get

Для обеспечения межпоточного взаимодействия допускается использование только pthread API.

```
#include <pthread.h>

class Value {
public:
    Value() : _value(0) { }

    void update(int value) {
        _value = value;
    }

    int get() const {
        return _value;
    }

private:
    int _value;
```

```
};

void* producer_routine(void* arg) {
  // Wait for consumer to start

  // Read data, loop through each value and update the value, notify
consumer, wait for consumer to process

}

void* consumer_routine(void* arg) {
  // notify about start
  // allocate value for result
  // for every update issued by producer, read the value and add to sum
  // return pointer to result
}

void* consumer_interruptor_routine(void* arg) {
  // wait for consumer to start

  // interrupt consumer while producer is running
}

int run_threads() {
  // start 3 threads and wait until they're done
  // return sum of update values seen by consumer

  return 0;
}

int main() {
    std::cout << run_threads() << std::endl;
    return 0;
}
```