Imię i nazwisko: ..... **F2** czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (2pkt)

Program oblicza ilość liczb pierwszych, wykorzystując wiele wątków. W aplikacji występuje wyścig. Proszę to wyeliminować dbając o skalowalność.

```
bool isPrime(long n) { //bada, czy n jest pierwsza
   if( n < 2 ) return false; // '0' and '1' is not prime
   if( n < 4 ) return true; // '2' and '3' is prime
   if( n % 2 == 0) return false;
   for(long i = 3; i * i <= n; i = i + 2)
      if(n % i == 0) return false;
   return true;
}
```

```
class ThreadPrime {
public:
   ThreadPrime(int n, atomic<int>& num) : n_max(n), num_primes(num) { }
   void operator()() {
      for(int i = 1; i < n_max; ++i ) {
         if(isPrime(i) )
            ++num_primes;
      }
   }
private:
   int n_max;
   atomic<int>& num_primes;
};

int main () {
   cout << "Program oblicza ilosc liczb pierwszych mniejszych niz N" << endl;
   cout << "podaj N:";
   int n_max = 0;
   cin >> n_max;

   atomic<int> num_primes = 0;

   ThreadPrime t1(n_max, num_primes);
   ThreadPrime t2(n_max, num_primes);
   ThreadPrime t3(n_max, num_primes);

   thread thrd1( ref(t1) );
   thread thrd2( ref(t2) );
   thread thrd3( ref(t3) );
   thrd1.join(); thrd2.join(); thrd3.join();
   cout << "num primes:" << num_primes << endl;
   cout << endl;
   return 0;
}
```

Pytanie 1 (0.5pkt)

Twoim zdaniem, które elementy przekazane na ZPR, są najbardziej przydatne?

Pytanie 2 (0.5pkt)

Twoim zdaniem, które elementy przekazane na ZPR, są najmniej przydatne?

Pytanie 3 (0.5pkt)

Jakie treści proponujesz dodać do ZPR w przyszłych edycjach?

Zadanie 3 (2.5pkt)
Program wysyła dane używając trzech gniazd TCP i synchronicznych blokujących operacji wysyłania ('Socket::send_sync'). Zmień implementację, aby wykorzystywać wysyłanie asynchroniczne nieblokujące ('Socket::send_async').

```
using Data = int; //nieistotne
using Error=const boost::system::error_code&;
using IOService = boost::asio::io_service;
using Event=function<void(Error)>;
class Socket { //obsluga wejscia/wyjscia - implementacja nieistotna
public:
   Socket(boost::asio::io_service&);
   void send_sync(Data d); //wysyla synchronicznie blokujaco, implementacja nieistotna;
   void send_async(Data d, Event event); //wysyla asynchronicznie, implementacja nieistotna;
   //inne metody nie sa istotne
};
```

```
struct Out {
   std::vector<Data> v; //nieistotne
   mutex m;

   Data pop() { //pobiera ostatni element z kolekcji i go usuwa.
      Data ret = 0;
      lock_guard<mutex> lock(m);
      if( ! v.empty() ) {
         ret = v.back();
         v.pop_back();
      }
      return ret;
   }
};

class Thread {
public:
   Thread(IOService& i, Out& o) : N(n), io(i), out(o) { }
   void operator()() {
      Socket s(io); //local socket
      while(1) {
         Data x = out.pop();
         if(x)
            s.send_sync(x);
         else
            break;
      }
   }
private:
   boost::asio::io_service& io;
   Out& out;
};

int main() {
   Out out;
   out.v = vector(100, 1313); //tworzenie paczki danych,
   //100 liczb o wartosci 1313, tego nie poprawiamy

   IOService io;

   Thread t1(1, io, out);
   Thread t2(1, io, out);
   Thread t3(1, io, out);

   io.run();

   std::thread thrd1( ref(t1) );
   std::thread thrd2( ref(t1) );
   std::thread thrd3( ref(t1) );

   thrd1.join(); thrd2.join(); thrd3.join();
   return 0;
}
```

Uwagi do prowadzącego (R.Nowak):

Imię i nazwisko:..... **F** czas: 60 min

Zadanie 4 (2pkt)

W pewnej grze gracze mogą zawierać transakcje. Każda transakcja może skończyć się dodaniem małego bonusu graczowi, który wydaje pieniądze. Kod nie działa poprawnie przy wywołaniu metody 'player1.take_all_money(player2)' – pieniądze zabierane są graczowi 2, ale nie są dodawane graczowi 1. Dodatkowo player2 nie otrzymuje bonusu (mimo, że powinien). Opisz błąd i popraw go korzystając z reguł borrow checkera Rust'a. Podpowiedź: analogiczny kod w Ruście skutkuje następującym błędem kompilacji: 'cannot borrow 'other' as mutable because it is also borrowed as immutable'. Załóż, że typ 'Money' ma przeciążone odp. operatory.

```
class PlayerAccount {
   public:
      PlayerAccount(Money money) : money(money) {};

      void take_all_money(PlayerAccount& other) {
         this->transaction(other, other.money);
      }

      void transaction(PlayerAccount& other, const Money& value) {
         if (other.money < value) {
            return;
         }
         other.money -= value;
         this->money += value;
         other.add_bonus_value(value);
      }

   private:
      int money;

      void add_bonus_value(const Money& transaction_value) {
         this->money += transaction_value * 0.01;
      }
};
```

Uwagi do prowadzącego (Ł.Neumann):

Zadanie 5 (1.5pkt)

Czy w C++ można napisać ekwiwalent Rust'owego mutex'a? Jeśli tak napisz w jaki sposób (może być kod/pseudokod/opis słowny), jeśli nie napisz dlaczego i podaj przykład niepoprawnego działania.

Pytanie 2 (0.5pkt)

Czego chciałbyś się dowiedzieć o RUST?