Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **24L kol 2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 1 (2pkt)

Proszę odpowiedzieć na poniższe pytania:

- Czy analiza statyczna ma przewagę nad dynamiczną? Czy któraś jest lepsza?

- Czy (i ewentualnie jak) można mierzyć jakość testu?

Zadanie 2 (2pkt)

Dana jest funkcja:

```
std::optional<Value> Service::method(const Other* obj, std::string_view name) {
  if ((name.size() == 0 )|| (obj == nullptr))
    throw InvalidArgument();
  if (const auto& item = this->cache[obj]; item.name == name)
     return item.value;
  return (obj == nullptr) ? std::nullopt : obj->valueOf(name);
}
```

Ile przypadków testowych należy napisać, by uzyskać:

- 100% pokrycia linii kodu,
- 100% pokrycia decyzji w kodzie.

Podaj elementy klas równoważności argumentów konieczne do uzyskania wspomnianych pokryć.

Zadanie 3 (2pkt)

Co jest nie tak z poniższym kodem?

```
int Client::upload(const Item& i, Request& r, bool ignored)
  // sprawdzamy połączenie
if (isConnected() && isAuthorized() &&
    (itemsQueue.size() == 0) && ((security == nullptr) || security->allowsUploads() || security->isAdmin()))
  {
    if (!r.header.topic.items.contain(Request::Header::Type::Active))
      return -2;
    auto tmp = r.body.items[3].data;

    if (tmp > 5) {
      tmp = 2036 * elementsCount(i);
      if (ignored)
        throw IncorrectRequest(r);
      switch (r.type) {
      case Type::Responding: return r.sendResponse(i, tmp);
      case Type::Blocked: return -3;
      default: return 0;
      }
    } else {
      if (ignored)
        throw IncorrectRequest(r);
      switch (r.type) {
      case Type::Responding: r.sendResponse(i, 15 * elementsCount(i) - 2);
      case Type::Blocked: return -3;
      default: return 0;
      }
   }
  }
  else
  {
    return -4;
  }
  return -5;
}
```

Proszę opisać jak należałoby poprawić ten kod (jakie operacje i w jaki sposób wykonane, kod nie jest konieczny).

Uwagi do prowadzącego (K. Grochowskiego):

Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **24L kol 2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 4 - współbieżne wzorce projektowe (2pkt)

Czy w programie pokazanym niżej występuje wyścig? [ ]

Jeżeli odpowiedź brzmi 'TAK' popraw kod w ramce, aby go wyelimionować.

```
using namespace std;
int main () {
    Queue q;
    Reader r1(q), r2(q); Writer w1(q), w2(q);
    thread t1(ref(r1)); thread t2(ref(r2)); thread t3(ref(w1)); thread t4(ref(w2));
    std::this_thread::sleep_for( chrono::milliseconds(2000));
    r1.finish(); r2.finish(); w1.finish(); w2.finish();
    t1.join(); t2.join(); t3.join(); t4.join();
    return 0;
}
```

Obiekty typu `Data` są duże. W kodzie występują obiekty `std::ref(x)`, są to obiekty pomocnicze, które zapobiegają tworzeniu kopii obiektu, więc np. `thread t(ref(p));` oznacza, że konstruktor `std::thread` nie wykona kopii obiektu `p`. Nie jest to kluczowe do znalezienia rozwiązania. Posługujemy się także listą dwukierunkową `std::list`, z operacjami `push_back` –wstawiania na koniec, `pop_front` – usuwanie początkowego elementu, `front` – zwraca początkowy element listy.

```
using Data = int; //Data to paczka danych do przetwarzania
const Data EMPTY = -1; //reprezentuje brak danych
struct Queue {
    std::list<Data> dane_;
    std::mutex mutex_;
    bool isEmpty() {
        std::lock_guard<std::mutex> lock(mutex_);
        return dane_.empty();
    }
    Data read() {
        Data out = EMPTY;
        if(!dane_.empty() ) {
            out = dane_.front();
            dane_.pop_front();
        }
        return out;
    }
    void write(Data d) {
        dane_.push_back(d);
    }
};
class Reader {
public:
    Reader(Queue& q) : finish_(false), queue_(q) {}
    void finish() { finish_ = true; }
    void operator()() {
        while(!finish_) {
            if(! queue_.isEmpty() ) {
              Data d = queue_.read();
              //ten komentarz oznacza algorytm, który wykonuje się długo i lokalnie, przetwarza 'd'
            }
        }
    }
private:
    Queue& queue_;
    volatile bool finish_;
};
class Writer {
public:
    Writer(Queue& q) : finish_(false), queue_(q) {}
    void finish() { finish_ = true; }
    void operator()() {
        while(!finish_) {
            //ten komentarz oznacza algorytm, który wykonuje się lokalnie i długo, na koniec inicjuje obiekt 'd'
            Data d;
            queue_.write(d);
        }
    }
private:
    Queue& queue_;
    volatile bool finish_;
};
```

Zadanie 5 - współbieżne wzorce projektowe (2pkt)

W obiektach typu `Reader`, `Writer`, których kod pokazano w Zadaniu 4 (przed Twoimi ewentualnymi zmianami) wykorzystano kolejkę pokazano poniżej, nie trzeba jej poprawiać. Funkcja `main` taka jak w Zadaniu 4.

```
using PNode = std::shared_ptr<Node>;
struct Node {
    Node(Data v, PNode n) : value(v), next(n) {}
    Data value;
    PNode next;
};
struct Queue {
    std::atomic<PNode> head_;
    bool isEmpty() {
        return head_.load() != nullptr;
    }
    void write(int val) {
        PNode new_node = make_shared<Node>( val, head_.load() );
        while (! head_.compare_exchange_weak(new_node->next, new_node))
            {}
    }
    Data read() {
        Data out = EMPTY;
        PNode ret_node = head_.load();
        while( ret_node && ! head_.compare_exchange_weak(ret_node, ret_node->next) )
            {}
        if( ret_node )
            out = ret_node->value;
        return out;
    }
};
```

- Czy to rozwiązanie jest wolne od wyścigów? ☐
- Czy to rozwiązanie wykorzystuje blokady/sekcje krytyczne? Proszę zaznaczyć poprawną odpowiedź i ewentualnie uzupełnić.
  - ☐ TAK
  - ☐ TAK, jeżeli (proszę dopisać) ☐
  - ☐ NIE
- Czy to rozwiązanie jest bardziej wydajne, niż rozwiązanie, które Ty zaproponowałaś/zaproponowałeś w Zadaniu 4 (rozwiązanie PO Twoich poprawkach)?
  - ☐ TAK
  - ☐ TAK, jeżeli (proszę dopisać) ☐
  - ☐ NIE

Uwagi do prowadzącego (R. Nowaka):