Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **25L kol 2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 1 (2pkt)

Proszę odpowiedzieć na poniższe pytania:

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

- Kiedy (zazwyczaj) nie używa się testów jednostkowych?

Zadanie 2 (2pkt)

Dana jest funkcja:

```
std::expected<Id, Error> Storage::save(Object* obj, std::optional<Id> hint) {
    if (obj == nullptr)
        return std::unexpected(Error::BadArg);
    if (obj->id().has_value() && hint.has_value())
        return std::unexpected(Error::Mismatch);
    const auto id = obj->id() ? *obj->id() : hint.value_or(defaultId);
    if (!serialize(id, obj))
        return std::unexpected(Error::Malformed);
    return id;
}
```

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?

```
#include <string>
struct Item {
 int x;
 int y;
 int w;
 int h;
 std::string name;
 auto area() { return w * h; }
};

int Screen::render(Item& i)
{
    if (i.h > 0)
    {
        if (i.w > 0) {
            auto x = i.area();
            if (x < textArea(i.name))
            {
                auto s = ellipsis(i.name);
                drawRect(i.x, i.y, i.w, i.h);
                return drawString(decorate(s));
            }
            else
            {
                drawRect(i.x, i.y, i.w, i.h);
                const auto d = decorate(i.name);
                auto r = drawString(d);
                if (r != 0)
                  return r;
                return 0;
            }
        }
        else {
            return -3;
        }
    }
    else
    {
        return -5;
    }
    return 0;
}
```

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: | | | | | | | | | | | | | | | | | | | | | | | | | | **25L kol 2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 4 - współbieżne wzorce projektowe (2pkt)

Czy w programie pokazanym niżej występuje wyścig? [ ]

```
int main () {
    Queue q;
    Reader r1(q), r2(q); Writer w1(q);
    thread t1(ref(r1)); thread t2(ref(r2)); thread t3(ref(w1));
    r1.finish(); r2.finish(); w1.finish();
    t1.join(); t2.join(); t3.join();
    return 0;
}
```

Popraw wydajność na platformach współbieżnych. Obiekty typu `Data` są duże. W kodzie występuje `std::ref(x)`, jest to udogodnienie, które zapobiega 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, zakladamy, ze obiekt jest 'duzy'
const Data EMPTY = -1; //reprezentuje brak danych
struct Queue {
    std::list<Data> dane_;
    std::mutex mutex_;
    bool isEmpty() {
        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_) {
        std::lock_guard<std::mutex> lock(queue_.mutex_);
        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_) {
        std::lock_guard<std::mutex> lock(queue_.mutex_);
        //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)

Uzupełnij tabelę, podaj możliwe wartości zwracane przez wymienione funkcje.
Kod poniżej wykorzystuje `std::ref(x)`, omówione w treści zad 4.

nazwa funkcji	możliwe zwracane wartości (oddziel przecinkiem)
`threads_free`	
`threads_atomic`	
`threads_mutex`	
`threads_mutex2`	

```
void serve_n(int i, int& counter) { counter += i; counter += i; }
void serve_a(int i, atomic<int>& counter) { counter += i; counter += i; }
void serve_m(int i, int& counter, mutex& m) {
    lock_guard<mutex> lock(m);
    counter += i; counter += i;
}
void serve_m2(int i, int& counter, mutex& m) {
    int local_counter = counter;
    {
      lock_guard<mutex> lock(m);
      local_counter += i; local_counter += i;
    }
    counter = local_counter;
}
int threads_free() {
    int counter = 0;
    thread th1( serve_n, 1, ref(counter) );
    thread th2( serve_n, 10, ref(counter) );
    th1.join(); th2.join();
    return counter;
}
int threads_atomic() {
    atomic<int> counter = 0;
    thread th1( serve_a, 1, ref(counter) );
    thread th2( serve_a, 10, ref(counter) );
    th1.join(); th2.join();
    return counter.load();
}
int threads_mutex() {
    int counter = 0;
    mutex m;
    thread th1( serve_m, 1, ref(counter), ref(m) );
    thread th2( serve_m, 10, ref(counter), ref(m) );
    th1.join(); th2.join();
    return counter;
}
int threads_mutex2() {
    int counter = 0;
    mutex m;
    thread th1( serve_m2, 1, ref(counter), ref(m) );
    thread th2( serve_m2, 10, ref(counter), ref(m) );
    th1.join(); th2.join();
    return counter;
}
```

Uwagi do prowadzącego (R. Nowaka):