Przykład z parzystością.

Wątek 1 inkrementuje zmienna globalną.

Wątek 2 ocenia parzystość tej zmiennej.

KOD7:

```cpp
#include <thread>
#include <cstdio>
#include <windows.h>

unsigned int counter = 0;

void increment(){
    for(;;){
        counter++;
        Sleep(2000);
    }
}

void parity(){
    for(;;){
        if (counter % 2){
            printf("%u jest nieparzyste\r\n", counter);
        }
        else{
            printf("%u jest parzyste\r\n", counter);
        }
        Sleep(2000);
    }
}

int main(){
    std::thread inc(increment);
    std::thread par(parity);

    inc.join();
    par.join();

    printf("Done\r\n");

    return 0;
}
```

```
1 jest nieparzyste
4 jest nieparzyste
5 jest nieparzyste
7 jest nieparzyste
8 jest nieparzyste
1 jest nieparzyste
2 jest parzyste
3 jest nieparzyste
4 jest parzyste
5 jest nieparzyste
7 jest nieparzyste
7 jest nieparzyste
9 jest nieparzyste
10 jest parzyste
```

**Jak widać wartości licznika nie zgadzają się. Często też nie zgadza się parzystość. Przykład ten równie dobrze może uruchomić się w sposób poprawny jednak trzeba pamiętać, że nie jest to pewne a oficjalna dokumentacja języka C++ określa wynik takiego kodu jako „undefined behavior”.**

Rozwiązaniem problemu jest mutex.

KOD8:

```cpp
#include <thread>
#include <cstdio>
#include <windows.h>
#include <mutex>

std::mutex counter_mutex;
unsigned int counter = 0;

void increment(){
    for(;;){
        counter_mutex.lock();
        counter++;
        counter_mutex.unlock();
        Sleep(2000);
    }
}

void parity(){
    for(;;){
        counter_mutex.lock();
        if (counter % 2){
            printf("%u jest nieparzyste\r\n", counter);
        }
        else{
            printf("%u jest parzyste\r\n", counter);
        }
        counter_mutex.unlock();
        Sleep(2000);
    }
}

int main(){
    std::thread inc(increment);
    std::thread par(parity);

    inc.join();
    par.join();

    printf("Done\r\n");

    return 0;
}
```

```
1 jest nieparzyste
2 jest parzyste
2 jest parzyste
4 jest parzyste
5 jest nieparzyste
6 jest parzyste
6 jest parzyste
8 jest parzyste
8 jest parzyste
10 jest parzyste
11 jest nieparzyste
12 jest parzyste
13 jest nieparzyste
14 jest parzyste
```

Po zastosowaniu mutexu wartości rosną w górę oraz wynik testu parzystości jest poprawny. Oczywiście mutex powoduje, że wątki muszą czekać jeden na drugi co wpływa na ogólny czas wykonania zadania.

Zadania do KOD7 oraz KOD8:

1. Przetestuj co stanie się gdy w jednym z wątków usunie się blokowanie i odblokowanie.
2. Dodaj w obu pętlach break po pewnej ilości iteracji po czym porównaj czasy wykoniania.

Zmienne lokalne wątku:

Przykład zastosowania zmiennych lokalnych wątku:

KOD9:

```cpp
#include <thread>
#include <cstdio>
#include <windows.h>

unsigned int counter = 0;

void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona się tylko raz mimo, że wątków jest więcej
    if(id == 1){
        printf("%u\n", counter);
    }
}

int main(){

    std::thread t1(increment,1);
    std::thread t2(increment,2);

    t1.join();
    t2.join();

    return 0;
}
```

```cpp
#include <thread>
#include <cstdio>
#include <windows.h>

thread_local unsigned int counter = 0;

void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona się tylko raz mimo, że wątków jest więcej
    if(id == 1){
        printf("%u\n", counter);
    }
}

int main(){

    std::thread t1(increment,1);
    std::thread t2(increment,2);

    t1.join();
    t2.join();

    return 0;
}
```

Po dopisaniu thread_local każdy wątek dostaje swoją kopię zmiennej counter.

Zadania do KOD9:

1. Zaalokuj tablice intów o rozmiarze 100, wypełnij ją losowymi liczbami z zakresu 1-10 i wypisz.

2. Zaalokuj 10 wątków i niech każdy z nich zsumuje komórki: [id*10;(id+1)*10] najpierw do zwykłej zmiennej a później do zmiennej thread_local.
3. Na końcu funkcji wątku wypisz: id -> wartość.