Семинар #1: Потоки. Домашнее задание.

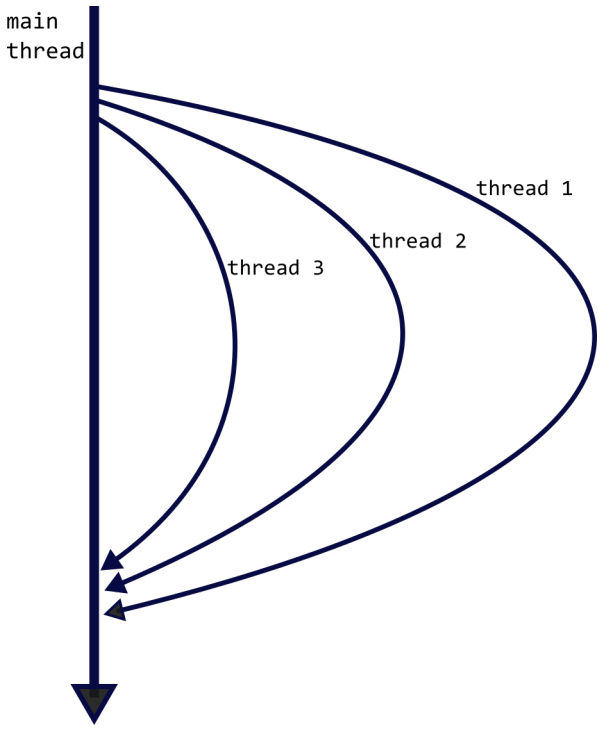
Задача 1. Запуск `n` потоков

Напишите программу, которая будет считывать целое число `n` из стандартного входа и создавать `n` новых потоков. `i`-й поток должен делать следующее:

- Писать сообщение: "Thread #i started."
- Ждать `i` секунд
- Писать сообщение: "Thread #i finished."

Для ожидания используйте функцию `sleep_for` из пространства имён `std::this_thread`. Пример использования этой функции:

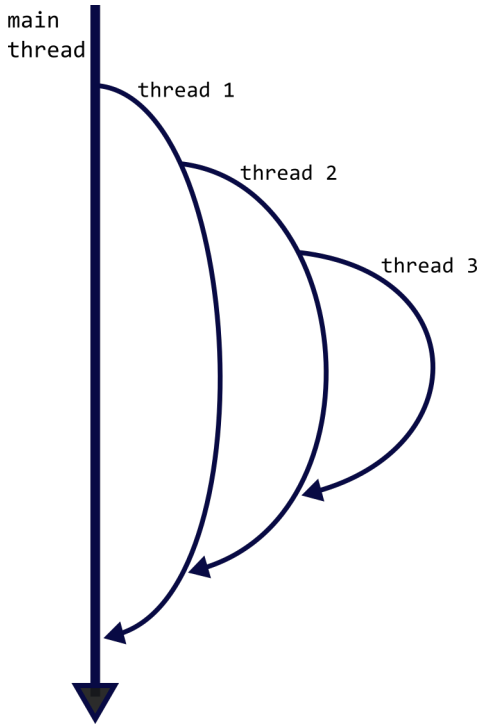
```
#include <iostream>
#include <thread>
#include <chrono>
using namespace std::chrono_literals;

int main()
{
    std::this_thread::sleep_for(1s);
    std::cout << "Waited for 1 second" << std::endl;

    std::this_thread::sleep_for(100ms);
    std::cout << "Waited for 100 millisecond" << std::endl;
}
```



Задача 2. Запуск потока в потоке

Напишите программу, которая будет считывать целое число `n` из стандартного входа и создавать `n` потоков. Но при этом главный поток (функция `main`) должен создать только первый поток. Второй поток должен создасться внутри первого. Третий – внутри второго. И так далее. Вообщем `i`-й поток должен:

- Писать сообщение: "Thread #i started."
- Ждать 200 миллисекунд
- Создавать `(i+1)`-й поток.
- Дожидаться завершения `(i+1)`-го потока с помощью метода `join`.
- Ждать 200 миллисекунд
- Писать сообщение: "Thread #i finished."



Задача 3. Поиск максимума

В файле `code/00problem_paralel_max.cpp` написана функция:

```
uint64_t getMax(const std::vector<uint64_t>& v)
```

которая принимает вектор чисел и возвращает максимальный элемент в этом векторе. Напишите функцию:

```
uint64_t getMax(int n, const std::vector<uint64_t>& v)
```

которая будет делать то же самое, но параллельно, используя `n` потоков. Протестируйте функцию, замерив скорость работы однопоточной и многопоточной версии.

Задача 4. Поиск максимума на диапазоне

В файле `code/01problem_paralel_max_iterators.cpp` написана шаблонная функция:

```
template <typename RandIt>
RandIt getMax(RandIt start, RandIt finish)
```

которая принимает два random-access итератора и находит максимальный элемент на диапазоне, задаваемом этими итераторами. Функция возвращает итератор на максимальный элемент. Напишите шаблонную функцию:

```
template <typename RandIt>
RandIt getMax(int n, RandIt start, RandIt finish)
```

которая будет делать то же самое, но параллельно, используя `n` потоков. Протестируйте функцию, замерив скорость работы однопоточной и многопоточной версии.

Задача 5. Параллельный `sort`

Напишите функцию:

```
template <typename RandIt, typename Comparator>
void parallelSort(int n, RandIt start, RandIt finish, Comparator comp)
```

которая будет сортировать диапазон, задаваемый итераторами `start` и `finish`, используя компаратор `comp`. Алгоритм должен работать в `n` потоков. Например, следующий вызов программы:

```
parallelSort(4, v.begin(), v.end(), [](int a, int b) {return a > b;});
```

должен сортировать вектор целых чисел `v` по убыванию, используя 4 потока. При реализации этой функции можно использовать однопоточную версию функции `std::sort`. Протестируйте функцию, замерив скорость работы однопоточной и многопоточной версии.

Задача 6. Повторить

Напишите функцию `iterate`, которая должна принимать на вход:

- целое число `n` – количество потоков
- функцию (в общем случае функциональный объект)
- аргументы передаваемой функции

Функция `iterate` должна `n` раз вызывать передаваемую функцию, каждый раз с новым аргументом. Каждый вызов должен происходить в отдельном потоке. Пример использования этой функции:

```
#include <iostream>
#include <thread>
#include <vector>
#include <chrono>
using namespace std::string_literals;

void func(const std::string& a, int b)
{
    std::cout << a << " " << b << std::endl;
}

// Тут нужно написать функцию iterate

int main()
{
    iterate(5, func, "Hello"s, 12345);
}
```

Этот код должен 5 раз выводить на экран `"Hello 12345"`. Выводы от разных потоков могут перемешиваться друг с другом. То есть на экран может вывестись, например, такое:

```
Hello Hello 12345
Hello 12345
Hello 12345
12345
Hello 12345
```