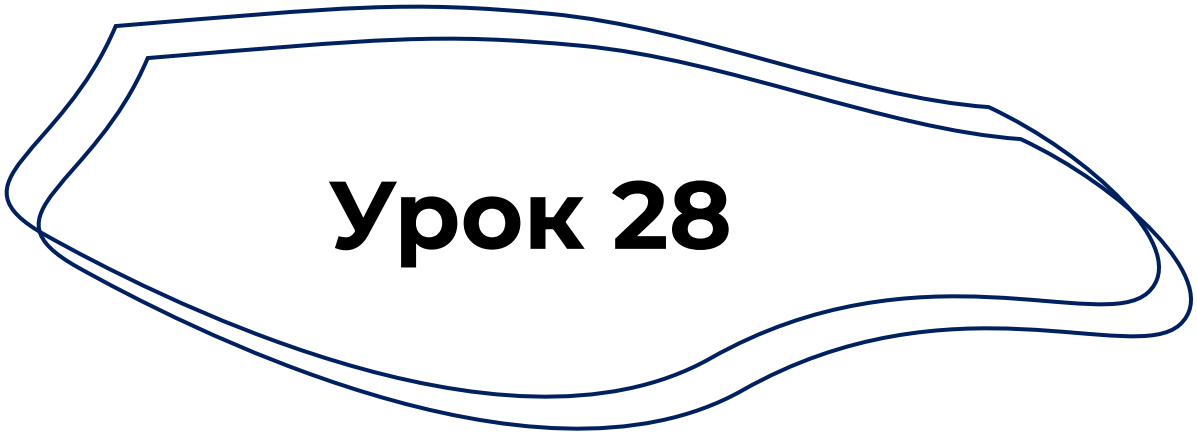
C++



Урок 28

Многопоточность ч 2

Задача

Задача: допишите реализацию потоковых вычислений и то, что они записывают на основе кода 28_001.cpp

```cpp
#include <iostream>
#include <thread>
#include <string>

template <typename T>
T get_pr(T val1, T val2)
{
    return val1 * val2;
}

using namespace std;
int main()
{

    // верните результат ф-ии get_pr в отдельном потоке

    int res; // целочисленный результат
    thread fi_th(/* */);

    double rez2; // вещественный результат
    thread se_th([/* */);

    char rez3; // символьный результат
    thread third_th(/* */);

    string rez4; // символьный результат
    auto f4 = [&]()
    {

    };
    thread four_th(/**/);

    /*
    дождаться завершения потока
    */

    cout << "Произведение целых чисел: " << res << endl;
    cout << "Произведение вещественнных чисел: " << rez2 << endl;
    cout << "Произведение символов: " << rez3 << endl;
    cout << "Произведение строк: " << rez4 << endl;
}
```

28_001.cpp

Задача

Задача: допишите реализацию класса фигура на основе кода 28_002.cpp

```cpp
template <typename T>
class Figure
{
    /*  поля + модификаторы
     */

    void show_sq()
    {
        cout << "Площадь равна: " << get_sq() << endl;
    }

    void show_per()
    {
        cout << "Периметр равен: " << get_per() << endl;
    }
};

int main()
{

    Figure<int> fig1(5, 10);

    thread fi_th(&Figure<int>::show_sq, fig1); // первый вариант передачи объекта и его метода в класс

    Figure<double> fig2(5.1, 10.1);
    thread sec_th([&]
                  { fig2.show_sq(); }); // второй вариант передачи объекта и его метода в класс (анонимная ф-ия)

    fi_th.join();
    sec_th.join();
}
```

28_002.cpp

Задача

Задача: допишите реализацию ф-ии main на основе кода 28_003.cpp

```cpp
class Figure
{
public:
    virtual double get_sq() = 0;
    virtual double get_per() = 0;

    virtual void show_sq() = 0;
    virtual void show_per() = 0;
};

template <typename T>
class square : public Figure
{
    T stor1, stor2;

public:
    square(T len1, T len2) : stor1(len1), stor2(len2) {}
    T get_sq()
    {
        return stor1 * stor2;
    }
    T get_per()
    {
        return 2 * (stor1 + stor2);
    }
    void show_sq()
    {
        cout << "Площадь прямоугольника равна: " << get_sq() << endl;
    }

    void show_per()
    {
        cout << "Периметр прямоугольника равен: " << get_per() << endl;
    }
};

template <typename T>
class triangle : public Figure
{
    T stor1, stor2, stor3;

public:
    triangle(T len1, T len2, T len3) : stor1(len1), stor2(len2), stor3(len3) {}
    T polu_per()
    {
        return (stor1 + stor2 + stor3) / 2.0;
    }
    T get_sq()
    {
        return sqrt(polu_per() * (polu_per() - stor1) * (polu_per() - stor2) * (polu_per() - stor3));
    }
    T get_per()
    {
        return stor1 + stor2 + stor3;
    }
    void show_sq()
    {
        cout << "Площадь треугольника равна: " << get_sq() << endl;
    }

    void show_per()
    {
        cout << "Периметр треугольника равен: " << get_per() << endl;
    }
};

int main()
{

    vector<Figure *> figures;
    Figure *temp;
    cout << "Какую фигуру надо добавить?\n0 - для завершения" << endl;
    string what;
    cin >> what;
    while (what != "0")
    {
        if (what == "кв")
        {
            temp = new square(3.4, 5.4);
            figures.push_back(temp);
        }
        else if (what == "треугольник")
        {
            temp = new triangle(3.1, 4.4, 5.35);
            figures.push_back(temp);
        }
        else
            cout << "Введено неверное значение, введите еще раз" << endl;

        cin >> what;
    }
    size_t col_elem = figures.size();

    // проходим с начала до половины абстрактного класса и выводим площадь

    // проходим с половины до конца абстрактного класса и выводим площадь

    // посчитаем суммарную площадь и периметр фигур

    sec_th.join();
    fi_th.join();
    third_th.join();
    fourth_th.join();

    cout << "суммарная площадь: " << summ_pl << "\nсуммарный периметр: " << summ_per << endl;

    // очистка памяти
}
```

28_003.cpp

Задача

Задача: используя многопоточность, очистите память.

Mutex

Mutex

Определение

Мьютекс («взаимное исключение») — это базовый механизм синхронизации. Он предназначен для организации взаимоисключающего доступа к общим данным для нескольких потоков с использованием барьеров памяти.

Идея

В программе наступает момент **барьерной синхронизации** (построение потоков). Для этого построения и нужен Mutex.

Mutex = регулировщик, который в определенный момент поднимает ключ и говорит ”стоять” остальным потокам. Как только поток завершил свое действие, он сообщает регулировщику, что остальные потоки могут продолжать.

Mutex – объект для синхронизации потоков.

Mutex

Мьютексы — это простейшие двоичные семафоры, которые могут находиться в одном из двух состояний — отмеченном или неотмеченном (открыт и закрыт соответственно).

Мьютекс отличается от семафора общего вида тем, что только владеющий им поток может его освободить, т.е. перевести в отмеченное состояние.

Mutex

Задачи

- В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом.
- Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

```cpp
#include <vector>
#include <iostream>
#include <mutex>
#include <thread>
#include <chrono>

std::mutex mutex;

void thread_func1(std::vector<int> &x)
{

    x.push_back(0);
}
void thread_func2(std::vector<int> &x)
{
    x.pop_back();
}

int main()
{
    std::vector<int> vec;
    std::thread th1([&]
                    { thread_func1(vec); });
    std::thread th2([&]
                    { thread_func2(vec); });

    for (auto &it : vec)
    {
        std::cout << it << " ";
    }
    th1.join();
    th2.join();
    return 0;
}
```

Какая проблема тут есть?

```cpp
#include <vector>
#include <iostream>
#include <mutex>
#include <thread>
#include <chrono>

std::mutex mutex;

void thread_func1(std::vector<int> &x)
{

    x.push_back(0);
}
void thread_func2(std::vector<int> &x)
{
    x.pop_back();
}

int main()
{
    std::vector<int> vec;
    std::thread th1([&]
                    { thread_func1(vec); });
    std::thread th2([&]
                    { thread_func2(vec); });

    for (auto &it : vec)
    {
        std::cout << it << " ";
    }
    th1.join();
    th2.join();
    return 0;
}
```

Есть шанс получить **ошибку сегментации**

Mutex

Основные действия:

- Объявление | `std::mutex mutex_name`;
- Захват мьютекса | `mutex_name.lock();`
  Поток запрашивает монопольное использование общих данных, защищаемых мьютексом. Дальше два варианта развития событий: происходит захват мьютекса этим потоком (и в этом случае ни один другой поток не сможет получить доступ к этим данным) или поток блокируется (если мьютекс уже захвачен другим потоком).
- Метод `try_lock` пытается получить права владения мьютексом без блокировки. Его возвращаемое значение можно преобразовать в bool и оно является true, если метод получает права владения; в противном случае — false.
- Освобождение мьютекса | `mutex_name.unlock();`
  Когда ресурс больше не нужен, текущий владелец должен вызвать функцию разблокирования unlock, чтобы и другие потоки могли получить доступ к этому ресурсу. Когда мьютекс освобождается, доступ предоставляется одному из ожидающих потоков.

Lock_guard

Lg

- — обертка
- конструктор вызывает метод lock для заданного объекта, а деструктор вызывает unlock
- в конструктор класса std::lock_guard можно передать аргумент std::adopt_lock - индикатор, означающий, что mutex уже заблокирован и блокировать его заново не надо
- std::lock_guard не содержит никаких других методов, его нельзя копировать, переносить или присваивать