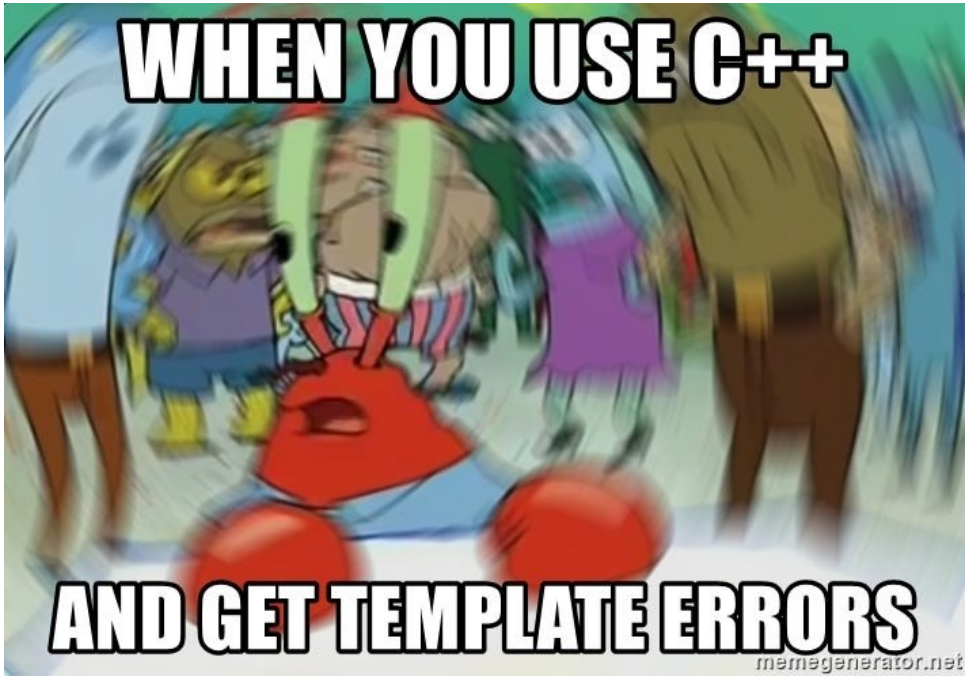
Семинар 5 по ООП

Енкапсулация, Предефиниране на потоци, Инициализиращ списък, Шаблон на клас



Енкапсулация (преговор)

- Един от 4-те принципа на ООП
- Не трябва да излагаме публично стейта на даден клас.
- Това се постига чрез подходящи access modifier-и.
- Public, private секция

Friend

- Причината поради която започнах със слайд за енкапсулация.
- Противно на името му, той не ни е приятел.
- В разрез с енкапсулацията.
- Прави всички полета на даден клас достъпни за дадена функция или друг клас, означени като приятелски
- Злоупотребата с него води до ниски резултати

Синтаксис

```
class Example
{
    private:
        int a,b;

    public:
        Example(int a, int b);

    friend int fun(Example ex);
    friend class Example2;
};
int fun(Example ex)
{
    return ex.a + ex.b;
}
```

```
class Example2
{
    private:
        int c;

    public:
        int sumWithExample(Example ex)
        {
            return ex.a + ex.b + c;
        }
        Example2(int c);

};
```

Лош пример

- Показаното на предния слайд илюстрира какво е friend, но и показва за какво не трябва да се използва!
- Показахме всички полета на класа Example на функцията fun и на класа Example2
- Сигнатурата трябва да е една и съща

Предефиниране на потоци

- Това е хубав пример за приложение на friend, което не е злоупотреба
- Нашата задача е да накараме std::cin и std::cout да заработят за обекти от нашия клас.
- Трябва да им дефинираме как да работят

Как се постига това

- Трябва да направим оператор, който приема поток и връща поток (така ще можем да стакваме).
- В този оператор ще дефинираме какво да прави при въвеждане / извеждане на класа ни.
- Входният поток има тип std::istream
- Изходния поток има тип std::ostream

Синтаксис

```
class Triangle
{
    private:
        int a, b, c;

    friend std::istream& operator>>(std::istream& in, Triangle& triangle);
    friend std::ostream& operator<<(std::ostream& out, const Triangle& triangle);
};

std::istream& operator>>(std::istream& in, Triangle& triangle)
{
    in >> triangle.a >> triangle.b >> triangle.c;
    return in;
}
std::ostream& operator<<(std::ostream& out, const Triangle& triangle)
{
    out << triangle.a << " " << triangle.b << " " << triangle.c;
    return out;
}
int main()
```

Инициализиращи списъци

- За да ни е по-лесно при правенето на конструктор, можем елементите, които не са динамична памет да ги инициализираме с инициализиращ списък.
- Също и конструкторите могат да се делегират

Пример

```
class Number
{
    private:
        int a;
    public:
        Number(int _a): a(_a){};
        Number(): Number(0) {};
};
```

Шаблон на клас

- Както има шаблони на функции, така има и на класове.
- Използва се, когато искаме да правим един и същ контейнер само с различни типове или искаме да пишем библиотека, която някой ще използва и типът с който ще използва нашия контейнер е предварително неизвестен нам.

Синтаксис

```
template <typename T>
class Vector
{
    private:
        size_t size;
        size_t capacity;
        T* arr;
}
```

```
    public:
        T output(T el)
        {
            std::cout << el;
            return el;
        }
```

Задачи

- Реализирайте шаблон на клас Vector, който представя динамичен масив от някакъв тип и има функции push, pop, оператор []!
- Реализирайте клас MyString, който представя динамичен низ. Нека има предефиниран оператор []. Оператор + и += за конкатенация на 2 низа от този тип и за добавяне на char[], потоци за вход и изход и функции substr и replace.