Отчет по 7ой лабе ассемблера

https://github.com/leeiozh/ass3sem/tree/master/7lab

1. (код 1-5 есть на гите) я создала класс Rocket и main вида

```
class Rocket {
private:
    double fuel_;
    float mass_;
public:
    bool is_work_;
    char name_;

    Rocket(double fuel, double mass) : fuel_(fuel), mass_(mass) {}

    double get_fuel() {
        return fuel_;
    }

    float get_mass() {
        return mass_;
    }

    void set_work(bool is_work) {
        is_work_ = is_work;
    }

    char set_name(char name){
        name_ = name;
        return name;
    }
};

int main() {
    Rocket rocket(10., 500.);
    rocket.set_work(true);
    rocket.set_name('Y');
    rocket.get_fuel();
    rocket.get_mass();
    return 0;
}
```

Скомпилила в .s, что-то получила, потом попереименовывала всё, закомментила половину и пришла к следующему:

имя функции Rocket(double, int, bool) будет сформированно вот так

_ZN6RocketC1Edib

имя функции Rocket::set_name(char) будет сформированно вот так

_ZN6Rocket8set_nameEc

имя функции main::Rocket::get_mass() будет сформированно как

_ZZ4mainEN6Rocket8get_massEv

коротко говоря, в начале всегда идет _ZN потом идет чиселка, равная количеству букв в первом слове названия функции, потом первое слово, затем вторая чиселка, равная количеству букв во втором слове, потом само второе слово, потом буква E, затем буквы, обозначающие тип аргумента (v – void, c – char, dd – 2 double, i – int, b – bool)
если перед нами конструктор, после имени добавляется C, если деструктор, то D

если класс объявлен внутри функции, то между первой Z и N появится Z9{func_name}E

2. Я написала в main

```
Rocket rocket(10., 500.);
Rocket rocket2(20., 100.);
rocket.set_work(true);
rocket2.set_name('Y');
```

и получила (самый важный кусок)

```
movsd  .LC0(%rip), %xmm0
movq   .LC1(%rip), %rdx
leaq   -48(%rbp), %rax
movapd         %xmm0, %xmm1
movq   %rdx, %xmm0
movq   %rax, %rdi
call   _ZN6RocketC1Edd
movsd  .LC0(%rip), %xmm0
movq   .LC1(%rip), %rdx
leaq   -32(%rbp), %rax
movapd         %xmm0, %xmm1
movq   %rdx, %xmm0
movq   %rax, %rdi
call   _ZN6RocketC1Edd
leaq   -48(%rbp), %rax
movl   $1, %esi
movq   %rax, %rdi
call   _ZN6Rocket8set_workEb
leaq   -32(%rbp), %rax
movl   $89, %esi
movq   %rax, %rdi
call   _ZN6Rocket8set_nameEc
```

и отсюда видно, что подобно тому, как мы работали со структурами, он под отдельные объекты класса выделяет отдельные куски памяти
здесь к rocket1 обращается в -48(%rbp), а к rocket2 в -32(%rbp)

3. заменю реализацию сеттера на this

```
void set_work(bool is_work) {
  this->is_work_ = is_work;
}
```

поменялось ничего (идентичные листинги)

заменю реализацию конструктора на this

```
Rocket(double fuel, double mass) : fuel_(fuel){
  this->mass_ = mass;
}
```

поменялось ничего (идентичные листинги)

вообще, *this - это указатель на конкретный объект класса, именно с которым в текущий момент работает метод и, наверное, от меня ожидался ответ по типу передачи в функцию указателя как в предыдущем пункте, например, -48(%rbp) или -32(%rbp)

на последней попытке попробую сравнить массы ракет функцией с this и без

```
bool who_fatter(Rocket& another){
   return mass_ > another.get_mass(); // return this->mass_ > another.get_mass();
}

Rocket rocket1(10., 500.);
Rocket rocket2(20., 501.);
rocket1.who_fatter(rocket2);
```

ну нет у них отличий, вероятно, необходимо придумать пример, в котором без this не обойтись, но, честно, я его не помню

4. я попробовала создать ракету от массы, объявленной в глобальных переменных, и от объявленной в локальных, кроме одной дополнительной передачи глобальной массы в функцию разницы не наблюдаю

я попробовала создать ракету вне main и внутри main
вовне ему пришлось создавать глобальную ракету, появились некие
.globl rocket1
_Z41__static_initialization_and_destruction_0ii
_GLOBAL__sub_I_rocket1
но он справился и сохранил указатель на нее, потом пользовался им как обычно
а внутри main все как обычно

для создания деструктора сделаю тип char *name_, чтобы потом в деструкторе освобождать память из-под него
получилось аналогично конструктору – для локальной ракеты он вызвался сразу в мейне после отработки функции, а деструктор глобальной ракеты подлетел после отработки мейна, вызовя _Z41__static_initialization_and_destruction_0ii

5. до этого у меня все методы были публичные, сделаем приватный очередной геттер

в названии особенностей не вижу, по реализации идентичны

после чуть более глубокой игры “найди отличие” ко мне закралась мысль, что ассемблер не видит разницы между приватными и пабликовыми полями и методами, и это сами плюсы меня бьют по рукам в случае неладного

6. (код есть отдельно на гите) сделала животного, хищника и кошку, результат ожидаем – конструкторы вызываются по очереди, т.е. из конструктора хищника вызывается *call* _ZN6AnimalC2Ecb, а из конструктора кошки *call* _ZN8PredatorC2Ev
в свою очередь поля налепляются друг рядом с другом как снежный ком, и инитятся по мере вызова контрукторов

7. полиморфизма не существует
на уровне ассемблера полиаморфные функции - это разные функции, которые могут как-то взаимодествовать друг с другом но из реализация под капотом в общем случае осуществляется независимо

8. (код есть отдельно на гите) я вернулась в ракете и сделала переменную is_work_ статичной

строчка листинга
movb %al, _ZN6Rocket8is_work_E(%rip)
почему-то навела меня на мысль, что он смотрит на статичные поля вообще как на глобальные переменные, но умеет отличать их от одного объекта к другому
в листинге мейна особенностей не вижу, только вот внутри реализации set_work такое нашла

почему статиковые методы не могут обращаться к нестатиковым полям – из-за разных областей видимости
статиковые методы это методы самого класса, они не могут вмешиваться в поля конкретного экземпляра (только если эти поля сами не являются статиковыми)
а если в терминах ассемблера – я не могу из глобальной функции вовне работать с локальными внутренними полями (как только эти поля станут видны извне, я смогу с ними работать)

9. (код есть отдельно на гите)
я перегрузила оператор < и сравнила две ракеты по массам
он откуда-то изнутри достал функцию `_ZN6RocketltES`
и каким-то не самым очевидным способом он просто сравнил массы ракет – первую массу он достал из текущего экземпляра, вторую он вызвал get_mass, а потом случилось

```
comiss-12(%rbp), %xmm0
seta    %al
leave
```

вызвается он непосредственно `calli _ZN6RocketltES_`

10. (код есть отдельно на гите)
я не стала заморачиваться и написала шаблонную функцию max

```
template <typename T>
T max(T a, T b)
{
   return (a > b) ? a : b;
}

int main(){
   int left = 14;
   int right = 55;
   int maximum = max(left, right);
   return 0;
}
```

на что он мне выдал `_Z3maxIiET_S0_S0_` с очевидной реализацией внутри (на eax)

на double он мне выдал `_Z3maxIdET_S0_S0_` с очевидной реализацией внутри (на xmm0)

на float он мне отдал `_Z3maxIfET_S0_S0_` с очевидной реализацией внутри (на xmm0)

на bool он мне отдал `_Z3maxIbET_S0_S0_` с реализацией на al и dl

думаю, с функциями все понятно, он просто подгоняет их сам в зависимости от требуемого типа переменных

для шаблона класса я чуть позаморочилась и написала простейший шаблонный массив
код есть, код понятен, для различных используемых типов он внутри себя независимо реализовывает их оболочки и пишет отдельные методы под каждый аргумент шаблона

11. (код есть) я сделала enum в виде цветов радуги
как и ожидалось, под капотом он реализован на интах, поэтому на операции сложения, к примеру, он реагирует нормально
enum это просто оболочка имен над законстанченными порядковыми интами