Lab 7

1-3. Methods and attributes

```
class My_Class {
public:
        int var_a = 53455345;
        void func(){ var_a = 9999999;}
};

int main() {
        My_Class obj_1;
        //std::cout << obj.a << "\n";
        obj_1.func();
        int b = obj_1.var_a;
        //std::cout << obj.a << "\n";
}
```

```
        .type   _ZN8My_Class4funcEv, @function
_ZN8My_Class4funcEv:
.LFB1522:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -8(%rbp)
        movq    -8(%rbp), %rax
        movl    $9999999, (%rax)
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1522:
        .size   _ZN8My_Class4funcEv, .-_ZN8My_Class4funcEv
        .text
        .globl  main
        .type   main, @function
main:
.LFB1523:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $53455345, -8(%rbp)
        leaq    -8(%rbp), %rax
        movq    %rax, %rdi
        call    _ZN8My_Class4funcEv
        movl    -8(%rbp), %eax
        movl    %eax, -4(%rbp)
        movl    $0, %eax
        leave
```

Classes are almost identical to structures – they stored exactly the same way(except when they have virtual members).

In this example class variable var_a is pushed into stack. Method func() is called as a usual function, but the name of it changes to _ZN8**My_Class**4**func**Ev to show that it belongs to a certain class.

%rdi is responsible for “this”: as we can see, the address of var_a in stack is transferred to %rdi before function call and it is used in function itself as a pointer to var_a.

5. Encapsulation

```
class My_Class {
private:
        int private_ = 1111111;
protected:
        int protected_ = 2222222;
public:
        int public_ = 333333;

        void my_print()
        {
                private_ = 10;
                protected_ = 20;
                public_ = 30;
        }
};

int main() {
        My_Class obj_1;
        //std::cout << obj.a << "\n";
        obj_1.my_print();
```

```
movl    $1111111, -12(%rbp)
movl    $2222222, -8(%rbp)
movl    $333333, -4(%rbp)
```

The compiler emits no code in relation to these access modifiers. It's purely a compile-time thing, it will remember if something is private and show an error message if you try to access it from the outside, but in the compiled code there's no concept of public or private or anything, it's just bytes.

And, of course, at the machine code level you can always do whatever you want. The only exception is when different access modifiers cause different optimizations to apply, so e.g. some member might disappear because the compiler knows it can never be accessed from the outside and it can simply rewrite expressions so it doesn't need it.

6. Inheritance

```
class Animal
{
public:
        int animal_legs = -10;
        int animal_tail = -10;
};

class Predator: public Animal
{
public:
        int predator_legs = -20;
        int predator_tail = -20;
};

class Bear: public Predator
{
public:
        int bear_legs = -30;
        int bear_tail = -30;
};

int main()
{
        Bear bear;
        bear.animal_legs = 4;
        bear.predator_legs = 10;
        bear.bear_legs = 100;

}
```

```
main:
.LFB1522:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $-10, -32(%rbp)
        movl    $-10, -28(%rbp)
        movl    $-20, -24(%rbp)
        movl    $-20, -20(%rbp)
        movl    $-30, -16(%rbp)
        movl    $-30, -12(%rbp)
        movl    $4, -32(%rbp)
        movl    $10, -24(%rbp)
        movl    $100, -16(%rbp)
```

Local variables from each class are initialized(added to stack) in the same order as they are placed in code.

7. Polymorphism

```
class Animal
{
public:
        int animal_var = 0;
        virtual void print(int a) {animal_var = a;}
};

class Predator: public Animal
{
public:
        int predator_var = 0;
        virtual void print(int a) {predator_var = 2 * a;}
};

int main()
{
        Predator predator;
        predator.print(10);
}
```

```
_ZN8Predator5printEi:
.LFB1523:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -8(%rbp)
        movl    %esi, -12(%rbp)
        movl    -12(%rbp), %eax
        leal    (%rax,%rax), %edx
        movq    -8(%rbp), %rax
        movl    %edx, 12(%rax)
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1523:
        .size   _ZN8Predator5printEi, .-_ZN8Predator5printEi
        .text
        .globl  main
        .type   main, @function
main:
.LFB1524:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        leaq    16+_ZTV8Predator(%rip), %rax
        movq    %rax, -32(%rbp)
        movl    $0, -24(%rbp)
        movl    $0, -20(%rbp)
        leaq    -32(%rbp), %rax
        movl    $10, %esi
        movq    %rax, %rdi
        call    _ZN8Predator5printEi
        movl    $0, %eax
        movq    -8(%rbp), %rdx
        xorq    %fs:40, %rdx
        je      .L4
        call    __stack_chk_fail@PLT
```

```
        .section        .data.rel.ro.local._ZTV8Predator,"awG",@progbits,_ZTV8
        .align 8
        .type   _ZTV8Predator, @object
        .size   _ZTV8Predator, 24
_ZTV8Predator:
        .quad   0
        .quad   _ZTI8Predator
        .quad   _ZN8Predator5printEi
        .weak   _ZTI8Predator
        .section        .data.rel.ro._ZTI8Predator,"awG",@progbits,_ZTI8Predat
        .align 8
        .type   _ZTI8Predator, @object
        .size   _ZTI8Predator, 24
_ZTI8Predator:
        .quad   _ZTVN10__cxxabiv120__si_class_type_infoE+16
        .quad   _ZTS8Predator
        .quad   _ZTI6Animal
        .weak   _ZTS8Predator
        .section        .rodata._ZTS8Predator,"aG",@progbits,_ZTS8Predator,com
        .align 8
        .type   _ZTS8Predator, @object
        .size   _ZTS8Predator, 10
_ZTS8Predator:
        .string "8Predator"
        .weak   _ZTI6Animal
        .section        .data.rel.ro._ZTI6Animal,"awG",@progbits,_ZTI6Animal,c
        .align 8
        .type   _ZTI6Animal, @object
        .size   _ZTI6Animal, 16
_ZTI6Animal:
        .quad   _ZTVN10__cxxabiv117__class_type_infoE+16
        .quad   _ZTS6Animal
        .weak   _ZTS6Animal
        .section        .rodata._ZTS6Animal,"aG",@progbits,_ZTS6Animal,comdat
        .align 8
        .type   _ZTS6Animal, @object
        .size   _ZTS6Animal, 8
_ZTS6Animal:
        .string "6Animal"
```

In case of using virtual functions we can notice, that class Predator is used as a global variable. Thus, besides the %rdi

register(“this”), each copy of the object has to carry around an extra pointer(in this case it’s %rax, which is added to stack).

8. Static

```
#include <iostream>

class My_class
{
public:
        static int Static_member;

};

//int My_class::Static_member = 10;

int main()
{
        My_class abba;
        abba.Static_member = 10;
        std::cout << abba.Static_member;
}
```

```
main:
.LFB1522:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $10, _ZN8My_class13Static_memberE(%rip)
        movl    _ZN8My_class13Static_memberE(%rip), %eax
```

Static member is used almost as global variables, except they have their own initialization and destruction:

```
_Z41__static_initialization_and_destruction_0ii:
.LFB2003:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        cmpl    $1, -4(%rbp)
        jne     .L5
        cmpl    $65535, -8(%rbp)
        jne     .L5
        leaq    _ZStL8__ioinit(%rip), %rdi
        call    _ZNSt8ios_base4InitC1Ev@PLT
        leaq    __dso_handle(%rip), %rdx
        leaq    _ZStL8__ioinit(%rip), %rsi
        movq    _ZNSt8ios_base4InitD1Ev@GOTPCREL(%rip), %rax
```

Static methods cannot access to non-static class members, because they refer to the class itself, not to the object, thus they don’t

interact with “this” pointer and don’t have access to object’s non-static variables.

9. Overriding

```
#include<iostream>

class Complex {
private:
        int real, imag;
public:
        Complex(int r, int i): real(r), imag(i){};
        Complex operator+(Complex const &obj) {
                Complex res(0, 0);
                res.real = real + obj.real;
                res.imag = imag + obj.imag;
                return res;
        }
        void print() { std::cout << real << " + i" << imag << '\n'; }
};

int main()
{
        Complex c1(10, 5);
        Complex c2(2, 4);
        Complex c3 = c1 + c2;
        c3.print();
}
```

Operator overriding is represented just as a function:

```
movq    %rdx, %rsi
movq    %rax, %rdi
call    _ZN7ComplexplERKS_
```

Function call in main

```
_ZN7ComplexplERKS_:
.LFB1525:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        leaq    -16(%rbp), %rax
        movl    $0, %edx
        movl    $0, %esi
        movq    %rax, %rdi
        call    _ZN7ComplexC1Eii
        movq    -24(%rbp), %rax
        movl    (%rax), %edx
        movq    -32(%rbp), %rax
        movl    (%rax), %eax
        addl    %edx, %eax
        movl    %eax, -16(%rbp)
        movq    -24(%rbp), %rax
        movl    4(%rax), %edx
        movq    -32(%rbp), %rax
        movl    4(%rax), %eax
        addl    %edx, %eax
        movl    %eax, -12(%rbp)
        movq    -16(%rbp), %rax
        movq    -8(%rbp), %rcx
        xorq    %fs:40, %rcx
```

10. Templates

```
#include <iostream>

template <typename T>
T sum(T a, T b)
{
        return a + b;
}

int main()
{
        int a = sum(2, 3);
        float b = sum(1.2, 3.4);
}
```

Two functions with the same realization, but with different names were created: one is referred to int `_Z3sumIiET_S0_S0_:`, other to float `_Z3sumIdET_S0_S0_:`.

11. Enum

```
enum planes{Boeing, Airbus, Embraer, Bombardier};

int main()
{
        planes plane_1 = Airbus;
        planes plane_2 = Boeing;
        std::cout << plane_2;
}
```

```
subq    $16, %rsp
movl    $1, -8(%rbp)
movl    $0, -4(%rbp)
```

Enum is not initialized at all.