

1.Как выглядят методы и поля классов.

Объявление класса:

```
_ZStL19piecewise_construct:
    .zero    1
    .local   _ZStL8__ioinit
    .comm    _ZStL8__ioinit,1,1
    .section .text._ZN14KolbasaSosiskaC2Ei,"axG",@progbits,_ZN14KolbasaSosiskaC5Ei,comdat
    .align 2
    .weak    _ZN14KolbasaSosiskaC2Ei
    .type    _ZN14KolbasaSosiskaC2Ei, @function
```

Конструктор:

```
_ZN14KolbasaSosiskaC2Ei:
.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)
    movq    -8(%rbp), %rax
    movl    -12(%rbp), %edx
    movl    %edx, (%rax)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Деструктор:

```
_ZN14KolbasaSosiskaD2Ev:
.LFB1532:
    .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)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
```

Как я отличил конструктор от деструктора? После названия конструкторов(у меня их несколько) C, после деструкторов D, после методов E

Поля класса такие:

```
_ZN14KolbasaSosiska9decrementEi:  _ZN14KolbasaSosiska9incrementEi:
.LFB1534:
```

Вызов методов.

```

leaq    -28(%rbp), %rax
movl    $97, %esi
movq    %rax, %rdi
call    _ZN14KolbasaSosiskaC1Ec
leaq    -36(%rbp), %rax
movl    $5, %esi
movq    %rax, %rdi
call    _ZN14KolbasaSosiska9incrementEi
leaq    -32(%rbp), %rax
movl    $3, %esi
movq    %rax, %rdi
call    _ZN14KolbasaSosiska9decrementEi
leaq    -28(%rbp), %rax
movl    $7, %esi
movq    %rax, %rdi
call    _ZN14KolbasaSosiska9incrementEi

```

Как этот метод, который обычная на вид функция, узнает, к полям какого объекта ему обращаться? В стеке локальных переменных данные классов лежат как массивчики с данными, поэтому он передает в функции(методы) указатель на то место в стеке, где лежит объект класса.

3. И что же такое this?

Это указатель на ячейку в стеке, в которой хранится первый элемент данных объекта(класса).

Код с обращением this → val

```

_ZN14KolbasaSosiskaC2Ev:
.LFB1526:
    .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
    movb    $83, (%rax)
    movq    -8(%rbp), %rax
    movl    $10, 4(%rax)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret

```

без this

```

.type    _ZN14KolbasaSosiskaC2Ev
_ZN14KolbasaSosiskaC2Ev:
.LFB1526:
    .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
    movb    $83, (%rax)
    movq    -8(%rbp), %rax
    movl    $10, 4(%rax)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc

```

```

void this_changer()
{
    cout<<this<<"\n"<<typeid(*this).name();
    KolbasaSosiska* A;
    A = this;
}

leaq    -48(%rbp), %rax
movq    %rax, %rdi
call    _ZN14KolbasaSosiska12this_changerEv

```

Хоть метод ничего не принимает, ему все же передается место в стеке, где лежат данные объекта через регистр %rdi.

```
_ZN14KolbasaSosiska12this_changerEv:
.LFB1536:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movq    %rdi, -24(%rbp)
    movq    -24(%rbp), %rax
    movq    %rax, -8(%rbp)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
```

4. Конструкторы / деструкторы.

Конструктор и деструктор локальных переменных(для функций, циклов, условий):

```
    leaq    -32(%rbp), %rax
    movl    $12, %esi
    movq    %rax, %rdi
    call    _ZN14KolbasaSosiskaC1Ei
    movl    $0, %ebx
    leaq    -32(%rbp), %rax
    movq    %rax, %rdi
    call    _ZN14KolbasaSosiskaD1Ev
```

Для глобальных немного сложнее:

```
_GLOBAL__sub_I_palka:
.LFB2019:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    $65535, %esi
    movl    $1, %edi
    call    _Z41__static_initialization_and_destruction_0ii
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
```

В блоке Global_sub_I_palka вызывается функция _Z41__static_initialization_and_destruction_0ii, которая вызывает и конструктор и деструктор объекта класса palka

```

_Z41__static_initialization_and_destruction_0ii:
.LFB2018:
    .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     .L8
    cmpl    $65535, -8(%rbp)
    jne     .L8
    leaq    _ZStL8__ioint(%rip), %rdi
    call    _ZNSt8ios_base4InitC1Ev@PLT
    leaq    __dso_handle(%rip), %rdx
    leaq    _ZStL8__ioint(%rip), %rsi
    movq    _ZNSt8ios_base4InitD1Ev@GOTPCREL(%rip), %rax
    movq    %rax, %rdi
    call    __cxa_atexit@PLT
    movl    $11, %esi
    leaq    palka(%rip), %rdi
    call    _ZN14KolbasaSosiskaC1Ei
    leaq    __dso_handle(%rip), %rdx
    leaq    palka(%rip), %rsi
    leaq    _ZN14KolbasaSosiskaD1Ev(%rip), %rdi
    call    __cxa_atexit@PLT
.L8:
    nop
    leave
    .cfi_def_cfa 7, 8
    ret

```

5. Инкапсуляция.

Как будут отличаться приватные и публичные поля и методы?

```

class KolbasaSosiska
{
protected:
    int protection = 88;
private:
    int privacy = 999;

public:
    int val = 1;
}

```

```

.cfi_def_cfa_register 6
movq    %rdi, -8(%rbp)
movq    -8(%rbp), %rax
movl    $88, (%rax)
movq    -8(%rbp), %rax
movl    $999, 4(%rax)
movq    -8(%rbp), %rax
movl    $1, 8(%rax)
nop

```

Private и protected поля выглядят одинаково. Также складываются вместе с public. Обращение к protected через методы тоже обыкновенное. Вызываю protected функцию в public protected_getting, все то же самое что и у public

```
protected:
    int protection = 88;
    void increment_protection()
    {
        protection += 100;
    }
}
```

Protected

Public

<pre> _ZN14KolbasaSosiska20increment_protectionEv: .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 (%rax), %eax leal 100(%rax), %edx movq -8(%rbp), %rax movl %edx, (%rax) nop popq %rbp .cfi_def_cfa 7, 8 ret .cfi_endproc </pre>	<pre> _ZN14KolbasaSosiska9incrementEv: .LFB1535: .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 8(%rax), %eax leal 100(%rax), %edx movq -8(%rbp), %rax movl %edx, 8(%rax) nop popq %rbp .cfi_def_cfa 7, 8 ret .cfi_endproc </pre>
---	---

Вот, что я заметил. Когда формируется название метода, между названиями класса и метода есть цифра. Не понятно зачем и для чего.

_ZN14KolbasaSosiska20increment_protectionEv:

~~Я сначала не обратил внимание, оказалось, что она совпадает для некоторых методов. Protected метод — цифра 20, Private — 15/18, Public — 9/7, Public вызывающий protected — 17.~~

Получается, public, private, protected не отличаются, либо ассемблер где-то спрятал(в флаге каком-нибудь), а я не нашёл.

```

class Animal
{
public:
    int body;
    int head;
    Animal()
    {
        body = 200;
        head = 100;
    }
    ~Animal(){}
    void growing()
    {
        body+=100;
        head+=50;
    }
};

class Predator: public Animal
{
public:
    int fangs;
    int claws;
    Predator()
    {
        fangs = 40;
        claws = 20;
    }
    ~Predator(){}
    void hunting()
    {
        fangs += 3;
        claws += 5;
    }
};

class Cat: public Predator
{
public:
    int tail;
    int mustache;
    Cat()
    {
        tail = 70;
        mustache = 30;
    }
    ~Cat(){}
    void meow()
    {
        tail += 10;
        mustache += 7;
    }
};

```

```

_ZN6AnimalC2Ev:
.LFB1543:
.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 $200, (%rax)
movq -8(%rbp), %rax
movl $100, 4(%rax)
nop
popq %rbp
.cfi_def_cfa 7, 8
ret

_ZN8PredatorC2Ev:
.LFB1550:
.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
movq %rdi, -8(%rbp)
movq -8(%rbp), %rax
movq %rax, %rdi
call _ZN6AnimalC2Ev
movq -8(%rbp), %rax
movl $40, 8(%rax)
movq -8(%rbp), %rax
movl $20, 12(%rax)
nop
leave
.cfi_def_cfa 7, 8
ret

_ZN3CatC2Ev:
.LFB1557:
.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
movq %rdi, -8(%rbp)
movq -8(%rbp), %rax
movq %rax, %rdi
call _ZN8PredatorC2Ev
movq -8(%rbp), %rax
movl $70, 16(%rax)
movq -8(%rbp), %rax
movl $30, 20(%rax)
nop
leave
.cfi_def_cfa 7, 8
ret

```

Конструкторы наследственных классов вызывают конструкторы родительских.

```

leaq -48(%rbp), %rax
movq %rax, %rdi
call _ZN6Animal7growingEv
leaq -48(%rbp), %rax
movq %rax, %rdi
call _ZN8Predator7huntingEv
leaq -48(%rbp), %rax
movq %rax, %rdi
call _ZN3Cat4meowEv

```

Для объекта класса методы родительских классов вызываются так же, как свои методы. Передаётся указатель на элемент и вызывается метод, независимо от того, родительский он или нет.

Полиморфизм(перегрузка методов) заключается только в том, что создаётся новая функция с другим названием. Например было Predator7hunting, при перегрузке в наследственном классе будет Cat7hunting.

Аналогично для функций, создаются доп.строки кода и меняется название немного

_Z8doublingi:

```
int doubling(int a)
{
    return 2*a;
}
int doubling(char a)
{
    return 0;
}
```

_Z8doublingc: