

Объектно-ориентированное программирование

Object-oriented programming

XXIV. Обзор курса

Course review

λ -выражения и архитектура фон Неймана

https://www.youtube.com/watch?v=vMDHpPN_p08



Для чего нужно высшее образование

<https://www.youtube.com/watch?v=SlasmhEcXDo>



Compiler optimizations



База от Дейкстры

“It is practically impossible to teach good programming to students that have had a prior exposure to **BASIC**: as potential programmers they are mentally mutilated beyond hope of regeneration.”

Edsger W. Dijkstra

Сначала был C

```
#include <string.h>
```

```
char * __cdecl strcpy(char * __restrict__ _Dest, const char * __restrict__ _Source);
```

```
char * __cdecl strcat(char * __restrict__ _Dest, const char * __restrict__ _Source);
```

```
int __cdecl strcmp(const char *_Str1, const char *_Str2);
```

```
size_t __cdecl strlen(const char *_Str);
```

```
_CRTIMP
```

```
char * __cdecl strdup(const char *_Src);
```

```
_CONST_RETURN
```

```
char * __cdecl strchr(const char *_Str, int _Val);
```

```
_CRTIMP
```

```
char * __cdecl strrev(char *_Str);
```

```
_CONST_RETURN
```

```
char * __cdecl strstr(const char *_Str, const char *_SubStr);
```

```
...
```

```
typedef char * str;
```

Инкапсуляция – инструмент абстракции

```
typedef struct {  
    char * text;  
    char * (*cons)(char*, const char*);  
} str;  
  
// ...  
  
// const char *cs1 = "hello";  
str s1 = { malloc(strlen(cs1) + 1), strcpy};  
s1.cons(s1.text, cs1);  
  
printf(s1.text);
```

Object-oriented C

```
typedef struct _class {
    size_t      size;
    void *      (*ctor)(void *self, va_list *app);
} Class;

typedef struct _string {
    const void * class;
    char *      text;
} String;

static void * String_ctor(void *_self, va_list *app);
void *      new(const void *_class, ...);
```

<https://www.cs.rit.edu/~ats/books/ooc.pdf>

Object-oriented C

```
static void *String_ctor(void *_self, va_list *app) {  
    String *self = _self;  
    const char *text = va_arg(*app, const char *);  
    self->text = malloc(strlen(text) + 1);  
    assert(self->text);  
    strcpy(self->text, text);  
    return self;  
}
```

* “stdarg.h”

Object-oriented C

```
void *new(const void *_class, ...) {  
    const Class *class = _class;  
    void *p = calloc(1, class->size);  
    assert(p);  
    *(const Class **)p = class;  
    if (class->ctor) {  
        va_list ap;  
        va_start(ap, _class);  
        p = class->ctor(p, &ap);  
        va_end(ap);  
    }  
    return p;  
}
```

Object-oriented C

```
static const Class _String = {  
    sizeof(String),  
    String_ctor  
};  
  
const void *string = &_String;  
  
void *a = new (string, "some text");
```

“If you are **"setting" values from the outside** of an object, you are doing *"simulated data structure programming"* **rather than object oriented programming.**”

Alan C. Kay

Классы

```
class str {  
    char * text{nullptr};  
public:  
    ~str();  
    str() = default;  
    str(const char*);  
    str(const str&);  
    str(const str&&) noexcept;  
    str &operator=(const str&);  
    str &operator=(str&&) noexcept;  
    void swap(str&);  
    // ...  
};
```

Классы

- ... группируют алгоритмы вместе с данными, которые они изменяют
- ... скрывают внутреннее состояние объекта от внешнего воздействия
- ... позволяют изменять поведение операторов других типов данных

Перегрузка

```
// str s = "0"; std::cout << 1 + s;  
  
class str {  
public:  
    friend std::ostream& operator<<(std::ostream&, const str&);  
    friend str operator+(int, const str&);  
};
```

Наследование

```
class pure_str : public str {  
public:  
    friend std::ostream& operator<<(std::ostream&, const pure_str&);  
  
    friend str operator+(int, const pure_str&) = delete;  
    // pure_str ps = "0"; std::cout << 1 + s;  
  
    friend str operator+(int, const str&) = delete;  
    // str s = "0"; std::cout << 1 + s;  
};
```


Композиция

```
class pure_str {  
    str string;  
public:  
    friend std::ostream& operator<<(std::ostream&, const pure_str&);  
  
    // friend str operator+(int, const pure_str&);  
    //      pure_str ps = "0"; std::cout << 1 + s;  
};
```

Другой пример

```
template <typename T> class iterator {  
protected:  
    T * ptr{nullptr};  
public:  
    iterator& operator++() { ptr++; return *this; }  
};  
  
template <typename T> class reverse_iterator : public iterator {  
public:  
    reverse_iterator& operator++() { ptr--; return *this; }  
};
```

Адаптер

```
template <typename T> class iterator {  
    T * ptr{nullptr};  
public:  
    iterator& operator++() { ptr++; return *this; }  
};  
  
template <typename Iterator> class reverse_iterator {  
    Iterator iter;  
public:  
    reverse_iterator& operator++() { iter--; return *this; }  
};
```

Полиморфизм

```
class str {  
    char * text{nullptr};  
public:  
    str(iterator<char>, iterator<char>);  
    // ...  
};  
  
str s1 = "hello";  
str s2(s1.end(), s1.begin());
```

Полиморфизм

```
template <typename T> class iterator;  
class reverse_iterator : public iterator<char>;  
  
// str s2(s1.end(), s1.begin())  
  
str::str(iterator<char> a, iterator<char> b) {  
    // ...  
    auto start = a; auto finish = b;  
    for (;start != finish;) {  
        *text++ = *start++;  
    }  
}
```

Динамический полиморфизм

```
template <typename T> class iterator;  
  
class reverse_iterator : public iterator<char>;  
  
str::str(iterator<char> &a, iterator<char> &b) {  
    // ...  
    auto start = a; auto finish = b;  
    for (;start != finish; *text++ = *start++);  
}  
  
str s2(reverse_iterator(s1.end()), reverse_iterator(s1.begin()));
```

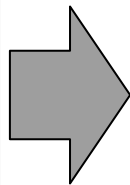
Статический полиморфизм

```
template <typename T> class iterator;  
  
template <typename Iterator> class reverse_iterator;  
  
template <typename Iter> str::str(Iter a, Iter b) {  
    // ...  
    auto start = a; auto finish = b;  
    for (;start != finish; *text++ = *start++);  
}  
  
str s2(reverse_iterator(s1.end()), reverse_iterator(s1.begin()));
```

Полиморфные классы

```
str s3 = U"🌲"; // UTF-8 -> char32_t*
```

```
template <typename T> class str {  
    T* text;  
public:  
    str(const T*);  
    // ...  
};
```



```
template<>  
class str<char32_t> {  
    char32_t * text;  
public:  
    str(const char32_t *);  
    // ...  
};
```


std::basic_string<>

```
namespace std {
    template <
        typename _CharT,
        typename _Traits,
        typename _Alloc>
    class basic_string {
        // ...

        // Use empty-base optimization:
        http://www.cantrip.org/emptyopt.html
        struct _Alloc_hider : allocator_type {
            pointer _M_p; // The actual data.
        };

        _Alloc_hider _M_dataplus;
        size_type _M_string_length;
```

```
enum {
    _S_local_capacity = 15 / sizeof(_CharT)
};

union {
    _CharT _M_local_buf[_S_local_capacity + 1];
    size_type _M_allocated_capacity;
};

// ...

};

typedef basic_string<char, ...> string;
};
```

https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/basic_string.h

“My guess is that *object-oriented programming* will be in the 1980s what *structured programming* was in the 1970s. **Everyone will be in favor of it.** Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (**differently**). And **no one will know just what it is.**”

T. Rentsch

Так что же такое ООП?

- **инкапсуляция** (абстракция состояния и избавление от манипуляции состоянием в коде, ~~public~~ и ~~private~~)
- **динамическое связывание** (дискриминированные объединения)
- **абстракция** (скрытая реализация, открытый интерфейс)
- **обобщенное поведение** (**generics**)

Так что же такое ООП?

Рассмотрим сферический ООП в вакууме...

Что такое ООР?

ООП – это идея **Lisp**, знающими людьми доведенная до абсолюта, позже подобранная посторонними людьми и доведенная до абсурда.

Я (2025)

Что такое OOP?

“You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.”

– Joe Armstrong, creator of Erlang



<https://www.youtube.com/watch?v=4xqkl953K6Y>



Почему такой подход?

“Everything we know about how people think and communicate is applicable. The mechanisms of **human thought and communication** have been engineered for millions of years, and we should **respect them** as being of sound design. Moreover, since **we must work with this** design for the next million years, it will save time if we **make our computer models compatible with the mind**, rather than the other way around.”

D. Ingalls

<https://www.cs.virginia.edu/~evans/cs655/readings/smalltalk.html>

Есть ли настоящий ООП в природе?

- Objective-C (message passing, dynamic resolution)
 - вместо виртуальных таблиц – “хэндлы” сообщений
 - объект выбирает какой метод вызывать в ответ на сообщение
- Erlang (actor model*)
 - каждый “процесс” (актер) содержит очередь сообщений, которые он обрабатывает
 - внутреннее “состояние” процесса меняется в ответ на сообщение
 - “let it crash”

* Модель была разработана Карлом Хьюэттом под влиянием объектов в Smalltalk. Smalltalk был разработан под влиянием системы PLANNER, разработанной Хьюэттом, которая стала основой языка Prolog. Prolog был основой разработки языка Erlang.

<https://softwareengineering.stackexchange.com/a/277469>

<https://www.cocoawithlove.com/2009/10/objective-c-niche-why-it-survives-in.html>

Actor model (за 27 минут!)

https://www.youtube.com/watch?v=vMDHpPN_p08



Еще немного о Smalltalk

<https://www.youtube.com/watch?v=mRwHZTNGdoY>

**Four Languages From
Forty Years Ago**

@ScottWlaschin