

Объектно-ориентированное программирование

Андрей Владимирович Попов

доцент кафедры анализа данных и ИИ

<https://andpop.ru>

МГУ им. Н.П. Огарева, 2025 год

Задачи курса

Задачи курса - теория

- Изучить основные понятия и концепции, связанные с ООП
- Рассмотреть несколько механизмов реализации ООП
 - ООП на классах (на примере PHP)
 - ООП на прототипах (на примере JavaScript)
 - Конвейерная обработка объектов (на примере PowerShell)
- Познакомиться с подходами и паттернами дизайна объектно-ориентированных приложений

Задачи курса - практика

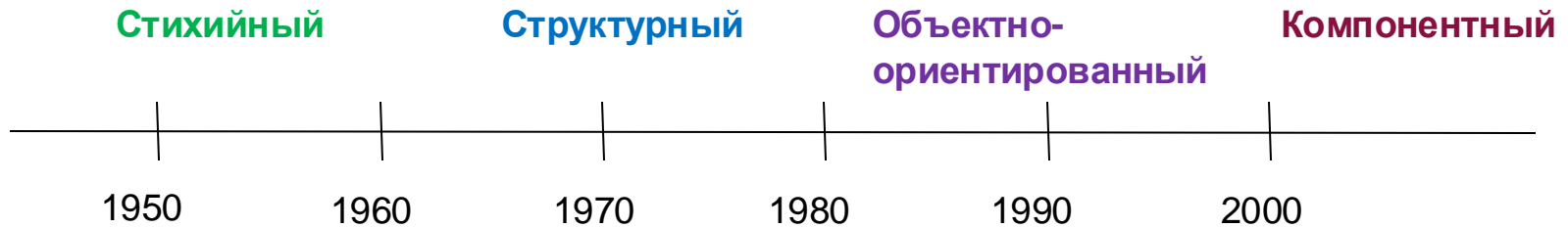
- Получить навыки работы с Git и GitHub
- Освоить инструменты экосистемы языков программирования (PHP и JavaScript)
- Написать код для решения предложенных задач

Лабораторные работы

- 8 работ
- Учебные репозитории: <https://github.com/andpop-mrsu>

1. Объектно-ориентированное программирование. Предыстория

Эволюция подходов к программированию



Цель: эффективное производство ПО требуемого качества

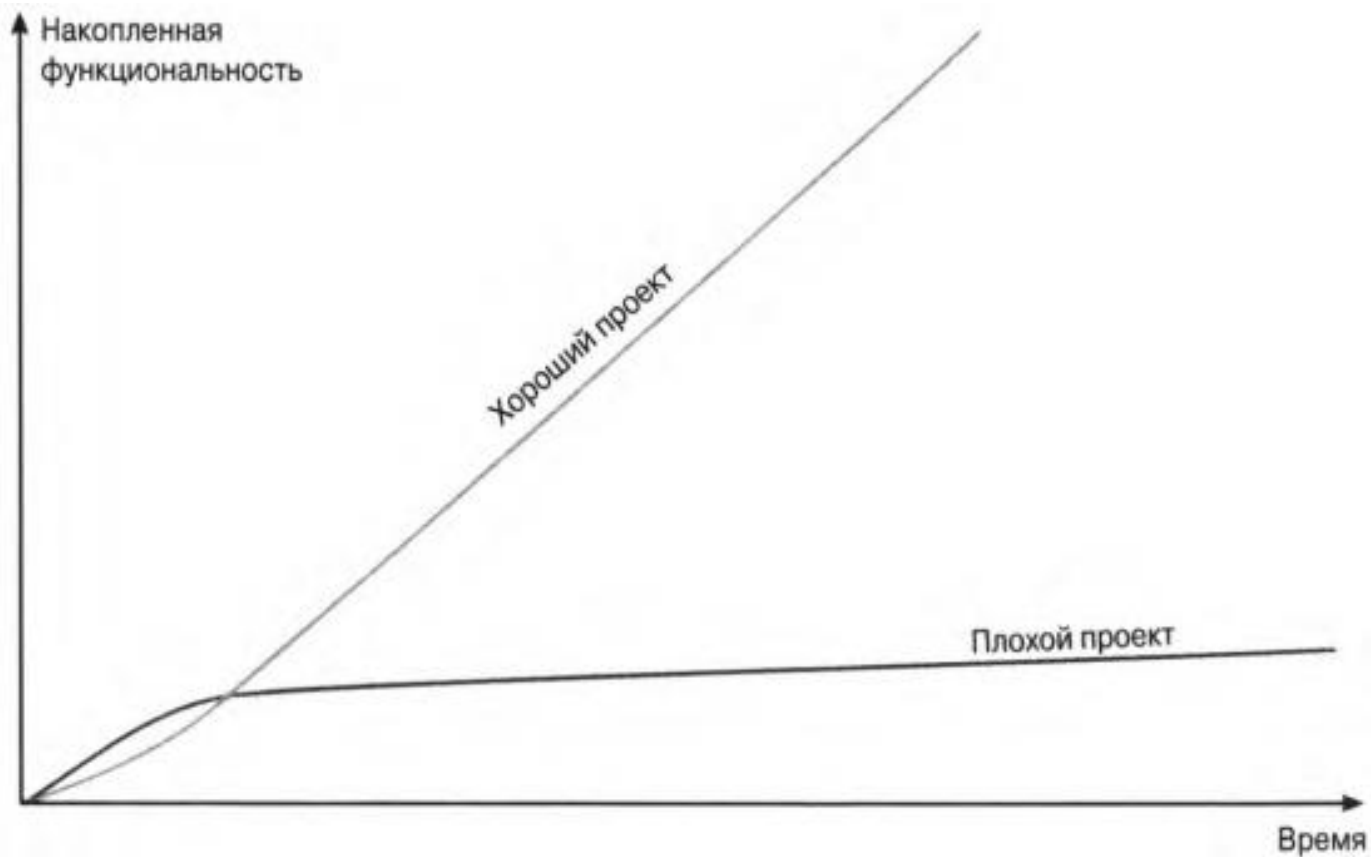
Жизненный цикл ПО

Написание
начальной версии

Изменения (поддержка, расширение)

Краткосрочная
производительность

Долгосрочная
производительность



Разница во внутреннем качестве ПО

Качество ПО

```
graph TD; A[Качество ПО] --> B[Внешние характеристики]; A --> C[Внутренние характеристики];
```

Внешние
характеристики

Для
пользователя
программы

Внутренние
характеристики

Для
разработчика
программы

Внутренние характеристики качества ПО

Каждый дурак может написать программу, которую может понять компьютер. Хороший программист пишет программу, которую может понять человек. Мартин Фаулер

Внутренние характеристики качества ПО

Каждый дурак может написать программу, которую может понять компьютер. Хороший программист пишет программу, которую может понять человек. Мартин Фаулер

- **Расширяемость и удобство сопровождения** — легкость изменения программной системы.
- **Гибкость и возможность повторного использования** частей системы в других системах.
- **Легкость чтения** исходного кода системы.
- **Интуитивная понятность** — легкость понимания системы и на уровне общей организации, и на детальном уровне отдельных операторов.
- **Слабая сопряженность** — компоненты ПО не находятся в сложной зависимости друг от друга.
- **Тестируемость** — возможная степень выполнения блочного и системного тестирования программы.

Главный ресурс программиста

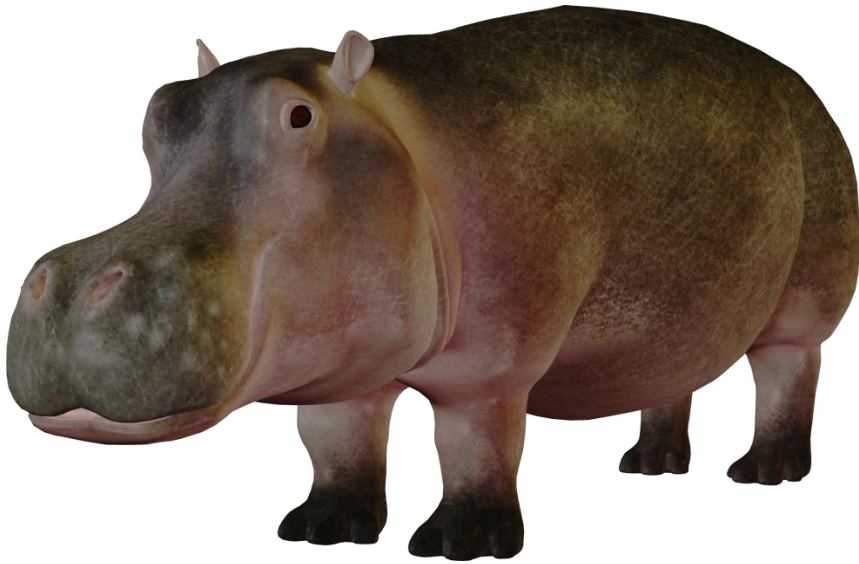
Концентрация внимания



Пример кода

Подходы к программированию

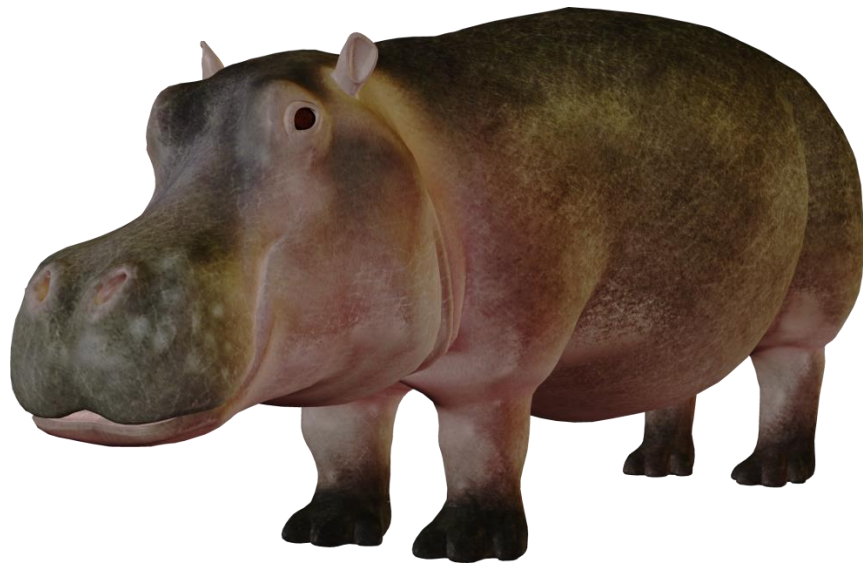
Задача – съесть бегемота



Сложная проблема

Подходы к программированию

Задача – съесть бегемота.



Сложная проблема

Простые фрагменты

Подходы к программированию

Задача – съесть бегемота.



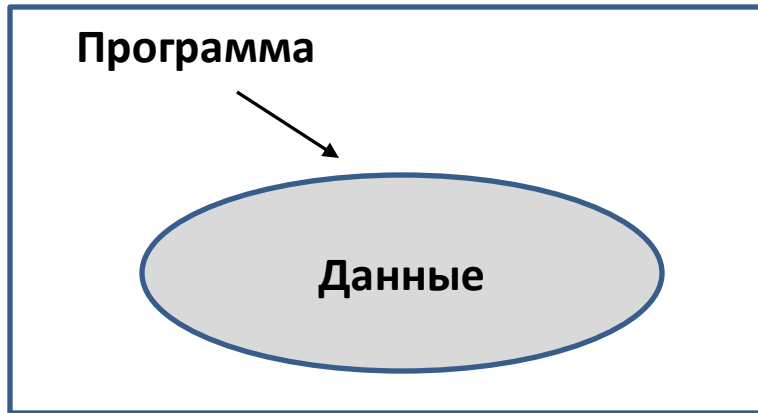
Сложная проблема

**Методика
проектирования ПО**



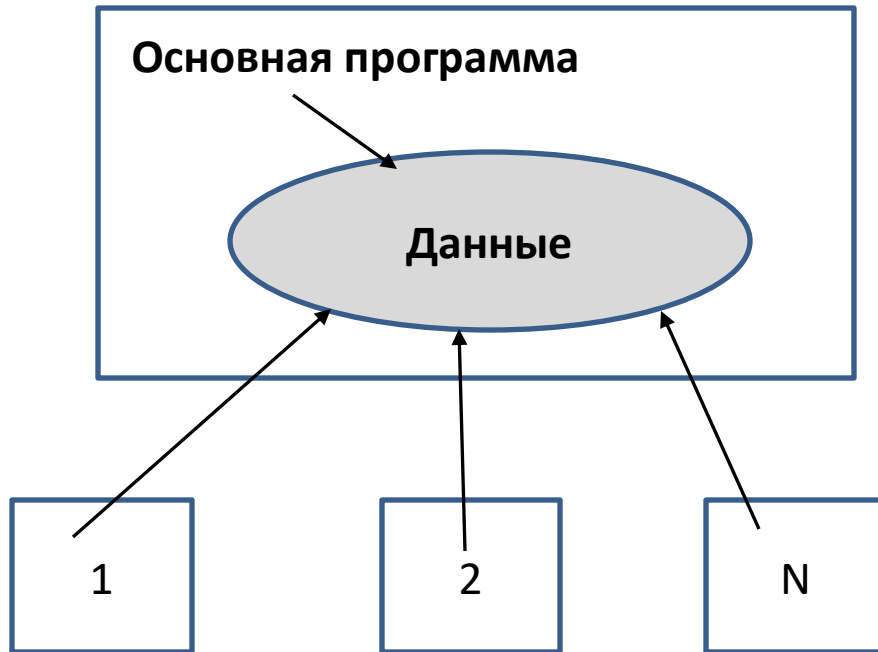
Простые фрагменты

Эволюция структуры программ



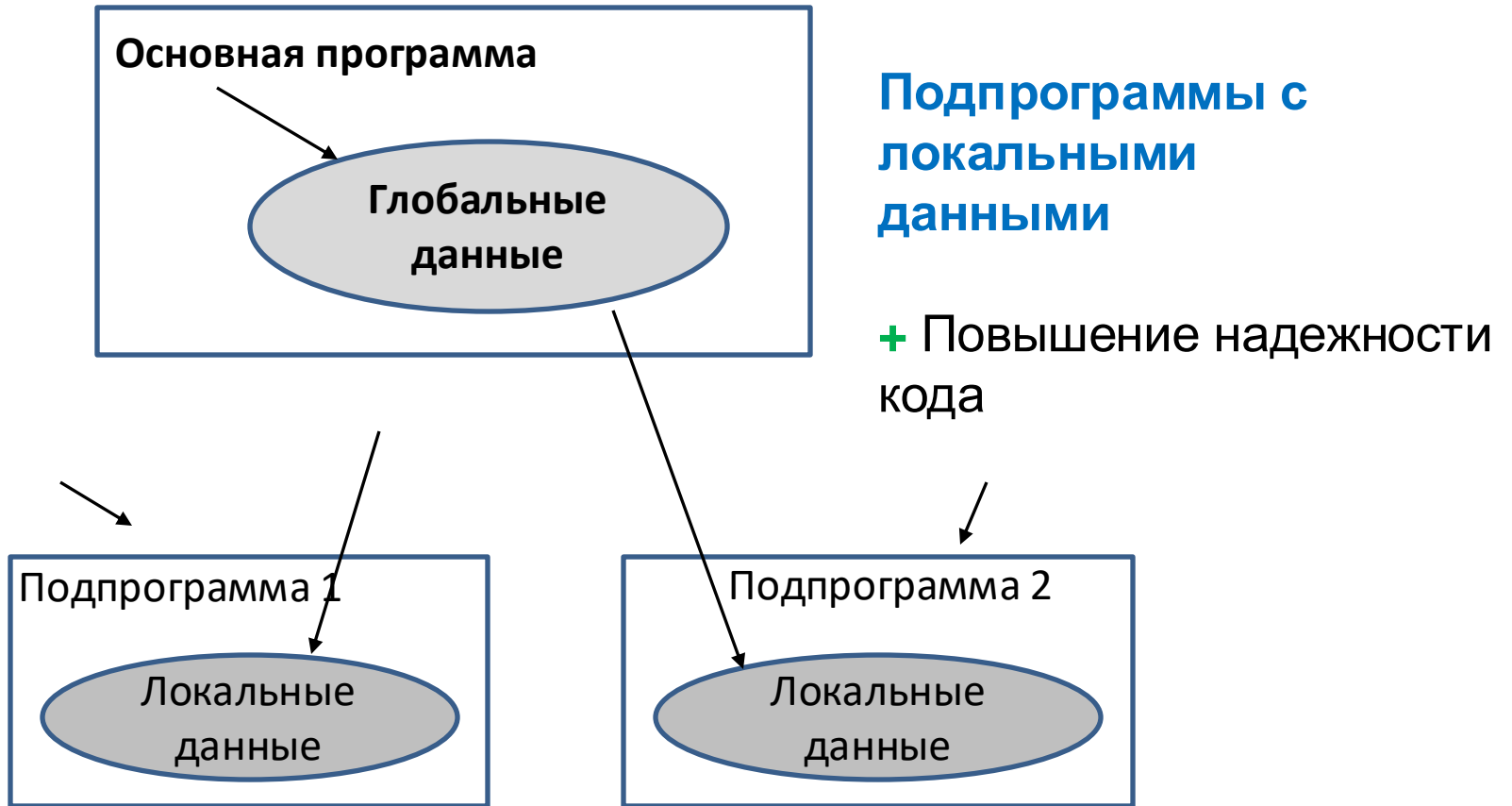
Монолитная структура

- Трудно отлаживать и повторно использовать код



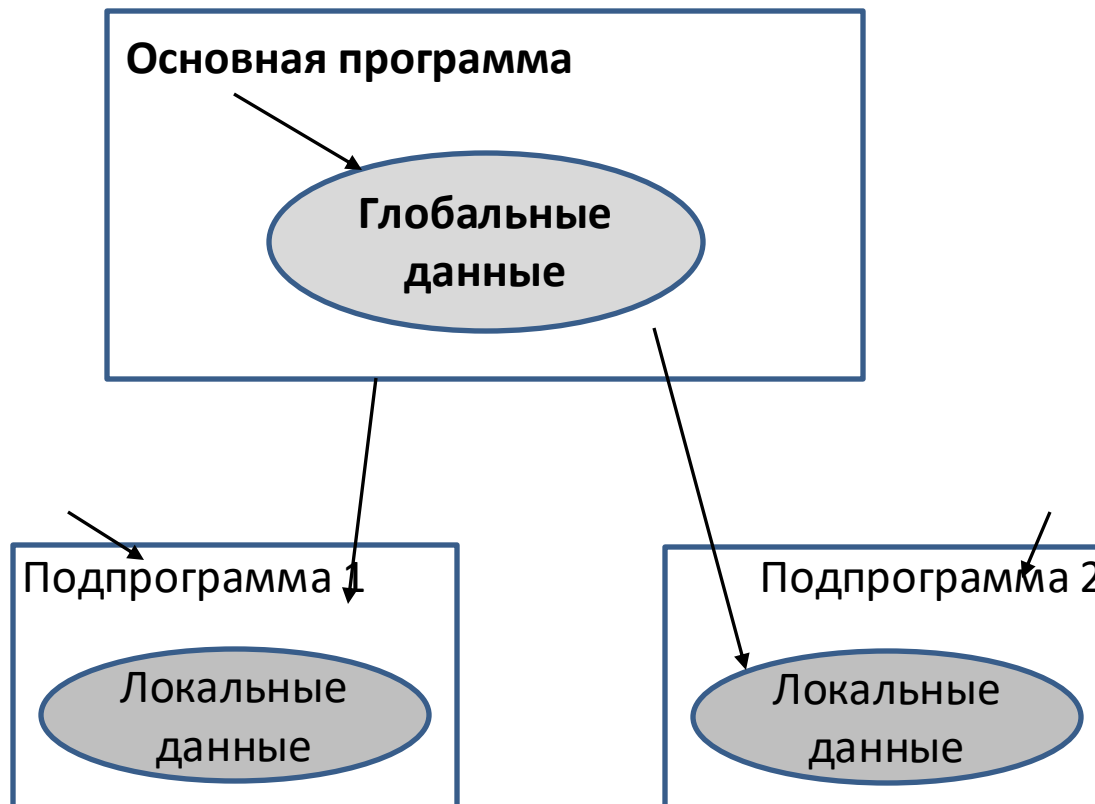
Подпрограммы, глобальные данные

- + Декомпозиция (упрощение)
- + Повторное использование кода
- Растет вероятность испортить данные



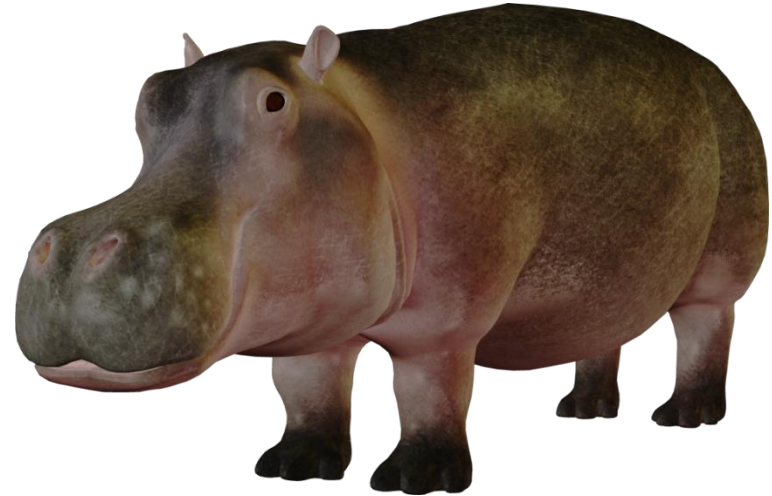
Процедурный подход

- Декомпозируй задачу
- Раздели код на процедуры.
- Используй наилучшие доступные алгоритмы.



Кризис стихийного программирования

Разработка «снизу-вверх» - вначале проектировали и реализовывали простые подпрограммы, из которых пытались строить сложную систему.



- Не было четких моделей для описания подпрограмм и методов их проектирования.
- Интерфейсы подпрограмм получались сложными.
- При сборке программы выявлялись ошибки, для их исправлений изменялись подпрограммы.

Кризис стихийного программирования

Разработка «снизу-вверх» - вначале проектировали и реализовывали простые подпрограммы, из которых пытались строить сложную систему.

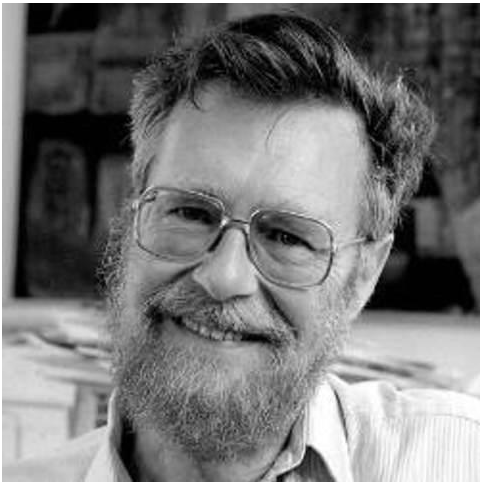


- Не было четких моделей для описания подпрограмм и методов их проектирования.
- Интерфейсы подпрограмм получались сложными.
- При сборке программы выявлялись ошибки, для их исправлений изменялись подпрограммы.

Требовалась технология, позволяющая создавать сложные программные продукты за приемлемое время.

Причина кризиса – запутанная структура программ.

Структурное программирование – методология разработки ПО для создания логически простых и понятных программ.



Эдсгер Дейкстра (1930-2002)

Статья «Structured programming» (1969 год) на конференции НАТО по разработке ПО



Никлаус Вирт (1934 г.р.)
Создатель языка Pascal

Структурное программирование

Три компонента:

1. Проектирование сверху вниз.
2. Структурное кодирование.
3. Модульное программирование.

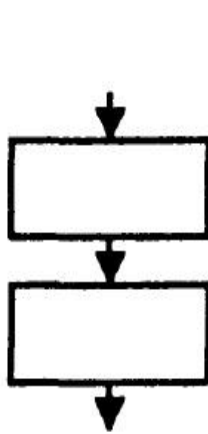
Проектирование «сверху вниз»



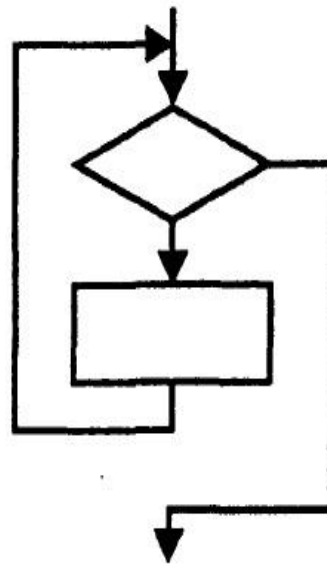
- Сначала напишите скелет программы на естественном языке.
- Вначале определение задачи в общих чертах, затем мелкие детали.
- Разрабатывайте тестовые данные заранее.
- Прежде чем начать программировать, разработайте проект.

Структурное кодирование: три базовые конструкции

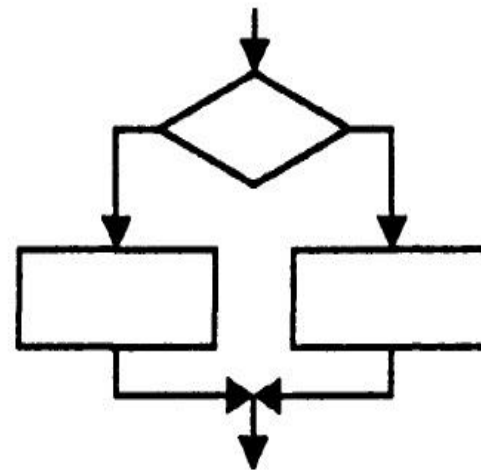
Любую программу можно написать с использованием только следующих логических структур:



Следование



Цикл



Ветвление

Базовые конструкции структурного программирования

Структурное кодирование: форматирование кода

```
/*! jQuery v1.12.2 | (c) jQuery Foundation | jquery.org/license */
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.
document?b(a,!0):function(a){if(!a.document)throw new Error("jQuery requires a window
with a document");return b(a)}:b(a)}("undefined"!==typeof window?window:this,function(a,b
){var c=[],d=a.document,e=c.slice,f=c.concat,g=c.push,h=c.indexOf,i={},j=i.toString,k=i.
hasOwnProperty,l={},m="1.12.2",n=function(a,b){return new n.fn.init(a,b)},o=
/^[\\s\\uFEFF\\xA0]+|[\\s\\uFEFF\\xA0]+$/g,p=/^-ms-/,q=-([\\da-z])/gi,r=function(a,b){return b.
toUpperCase()};n.fn=n.prototype={jquery:m,constructor:n,selector:"",length:0,toArray:
function(){return e.call(this)},get:function(a){return null!=a?0>a?this[a+this.length]:
this[a]:e.call(this)},pushStack:function(a){var b=n.merge(this.constructor(),a);return b.
prevObject=this,b.context=this.context,b},each:function(a){return n.each(this,a)},map:
function(a){return this.pushStack(n.map(this,function(b,c){return a.call(b,c,b)}))},slice:
function(){return this.pushStack(e.apply(this,arguments))},first:function(){return this.
eq(0)},last:function(){return this.eq(-1)},eq:function(a){var b=this.length,c=+a+(0>a?b:0
);return this.pushStack(c>=0&&b>c?[this[c]]:[])},end:function(){return this.prevObject||
this.constructor()},push:g,sort:c.sort,splice:c.splice},n.extend=n.fn.extend=function(){
var a,b,c,d,e,f,g=arguments[0]||{},h=1,i=arguments.length,j=!1;for("boolean"==typeof g&&(
j=g,g=arguments[h]||{},h++),"object"==typeof g||n.isFunction(g)||g==={}),h===i&&(g=this,h
--);i>h;h++)if(null!=(e=arguments[h]))for(d in e)a=g[d],c=e[d],g!==c&&(j&&c&&(n.
isPlainObject(c)||b=n.isArray(c)))?(b?(b=!1,f=a&&n.isArray(a)?a:[]):f=a&&n.isPlainObject
(a)?a: {},g[d]=n.extend(j,f,c)):void 0!==c&&(g[d]=c));return g},n.extend({expando:"jQuery"
+(m+Math.random()).replace(/\\D/g,""),isReady:!0,error:function(a){throw new Error(a)},
noop:function(){},isFunction:function(a){return"function"===n.type(a)},isArray:Array.
isArray||function(a){return"array"===n.type(a)},isWindow:function(a){return null!=a&&a==a
.window},isNumeric:function(a){var b=a&&a.toString();return!n.isArray(a)&&b-parseFloat(b
)+1>=0},isEmptyObject:function(a){var b;for(b in a)return!1;return!0},isPlainObject:
function(a){var b;if(!a||"object"!==n.type(a)||a.nodeType||n.isWindow(a))return!1;try{if(
a.constructor&&!k.call(a,"constructor")&&!k.call(a.constructor.prototype,"isPrototypeOf"
```

Структурное кодирование: форматирование кода

- Визуализация логической структуры программы с помощью **отступов**.
- Определенный **стиль форматирования и именования**.
- Ограничение длины выражений, дополнительные пробелы для операндов и пустые строки для выделения программных блоков.
- **Осмысленные названия** переменных – код должен быть самодокументируемым и понятным даже без комментариев.
- Комментарии должны пояснять цель и задачу кода, а не ее решение.
 - *«Получение информации о текущем сотруднике»* - комментарий цели.
 - *«Обновление объекта employeeRecord»* – комментарий в терминах решения проблемы

Процедурный подход

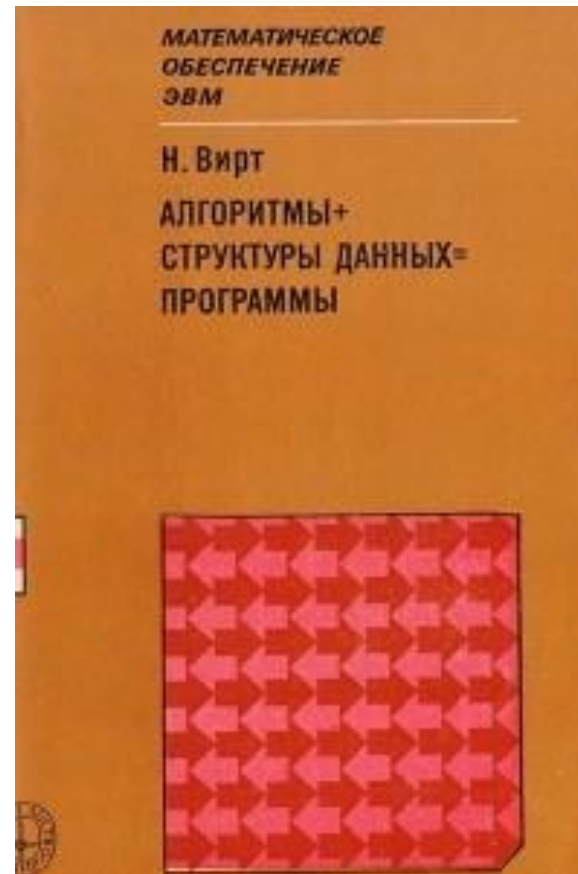
- Тип переменной определяет множество ее допустимых значений.
- Переменные (данные) могут иметь внутреннюю структуру (структуры, записи, словари, ассоциативные массивы и т.п.).

Type

```
Person = record
    Name: string;
    Address: string;
    Age: integer;
end;
```

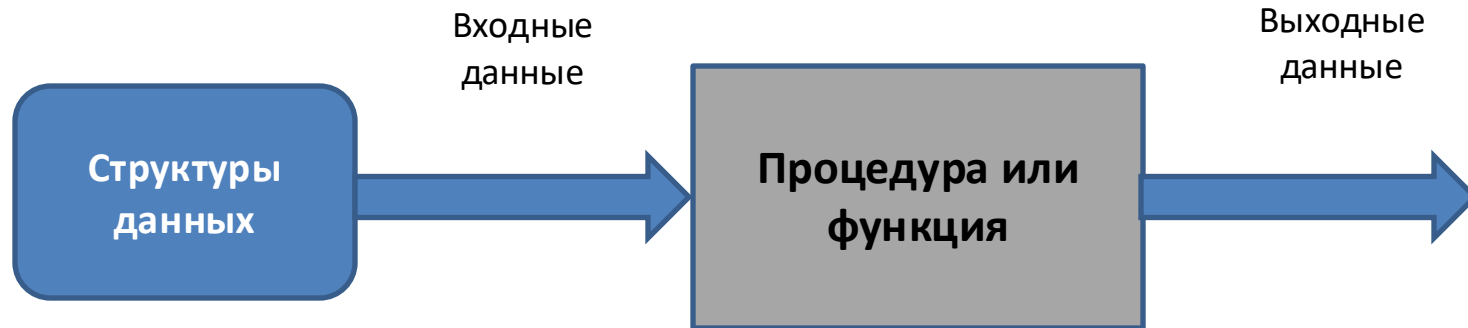
Процедурный подход

- Приложение – это последовательность операций (алгоритмы и бизнес-логика), воздействующих на отдельные структуры данных.



Процедурный подход

- Сложные программы делятся на простые фрагменты с помощью подпрограмм.
- Данные и операции – логически разные сущности.
 - Данные составляют состояние системы
 - Алгоритмы определяют её поведение.



Процедурный подход: создаем данные

- Ручное создание структуры данных

```
$profile = [  
    'id' => 42,  
    'name' => 'Vasya',  
    'emails' => [  
        'vasya@examlpe.com'  
    ]  
];
```

- Функция-фабрика для создания новых экземпляров

```
function newProfile(int $id, string $name, string $email): array  
{  
    return [  
        'id' => $id,  
        'name' => $name,  
        'emails' => [$email]  
    ];  
}  
  
$profile = newProfile(42, 'Vasya', 'vasya@examlpe.com');
```

Процедурный подход: функции для обработки данных

- Процедура для добавления нового e-mail с проверкой

```
function addEmail(array &$profile, string $email): void
{
    if (!in_array($email, $profile['emails'])) {
        $profile['emails'][] = $email;
    }
}
```

```
addEmail(&$profile, 'new@site.com');
```

Структура данных передается в функцию по ссылке, т.е функция изменяет исходное состояние.

Функциональный подход

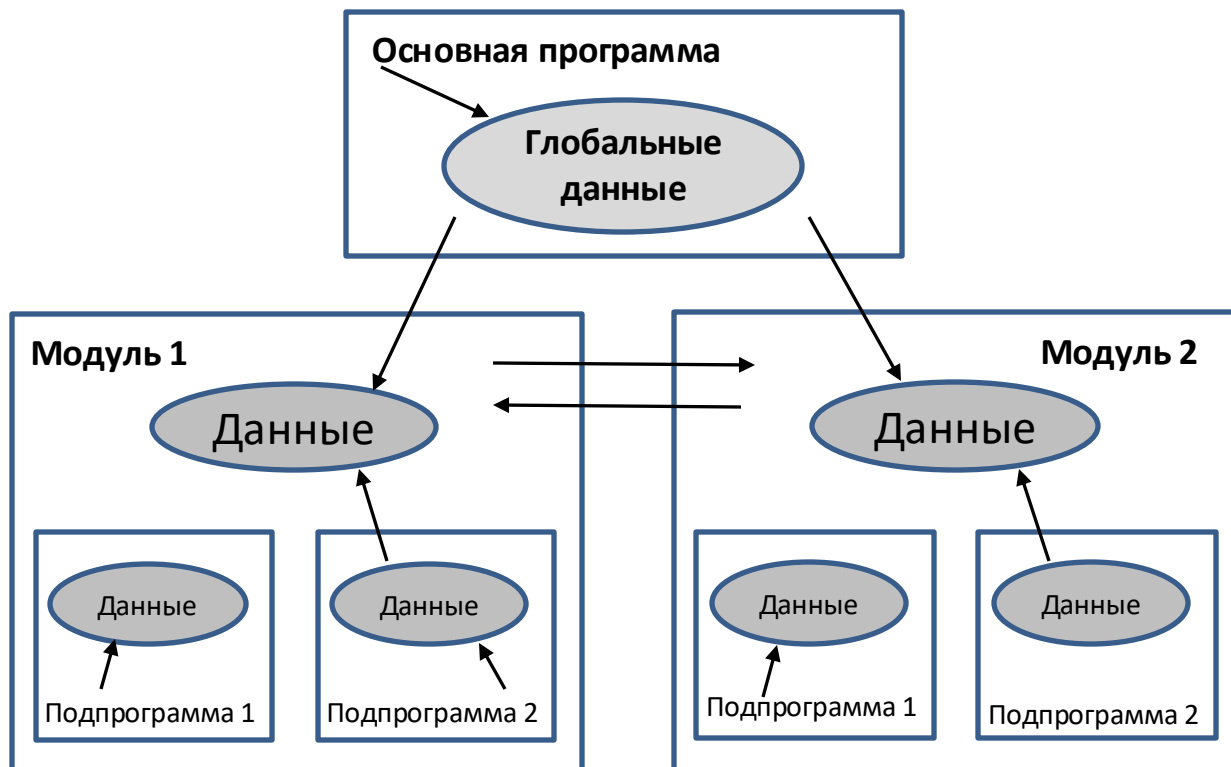
- Функция на основании исходного профиля вычисляет новое состояние системы (новый профиль).

```
function addEmail(array $profile, string $email): array
{
    return
        !in_array($email, $profile['emails'])
            ? [
                'id' => $profile['id'],
                'name' => $profile['name'],
                'emails' => [ ...$profile['emails'], $email ]
            ]
            : $profile;
}
```

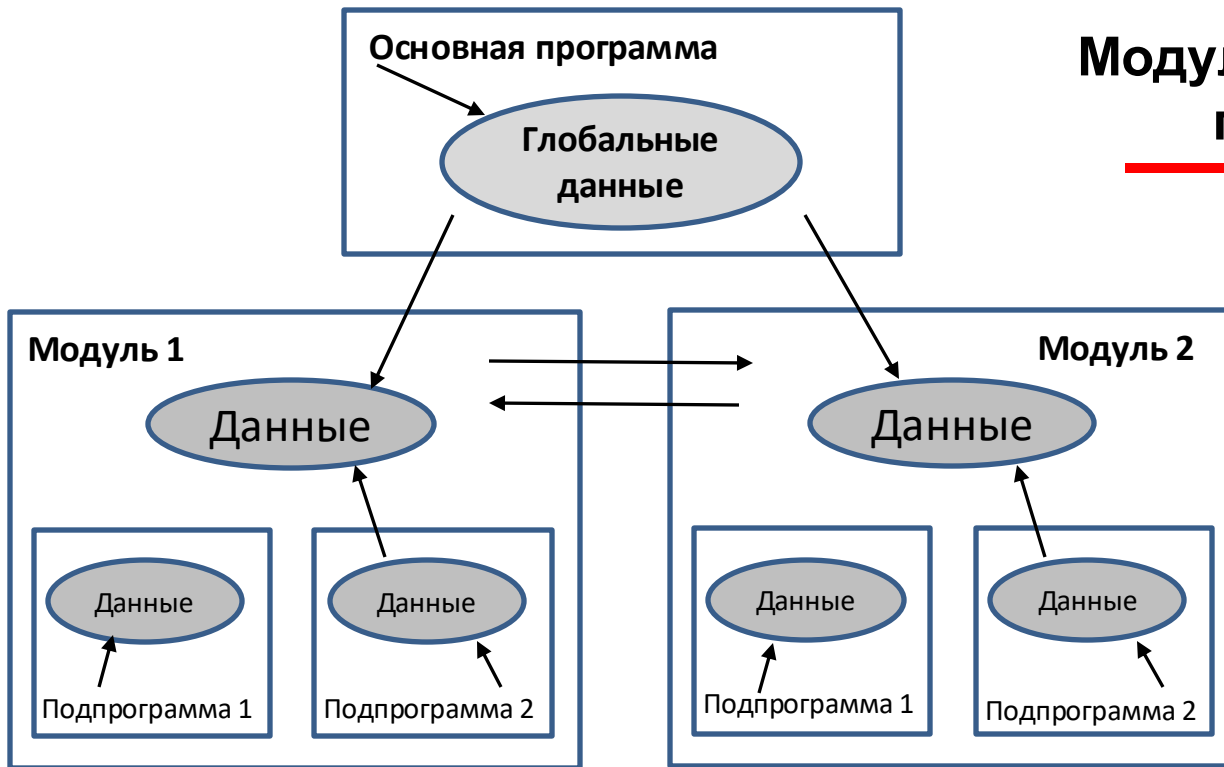
```
$newProfile = addEmail($oldProfile, 'new@site.com');
```

Модульный подход: инкапсуляция и сокрытие данных

Данные и обрабатывающие их функции локализуются (инкапсулируются) в модулях и защищаются от внешних воздействий с помощью интерфейсов.

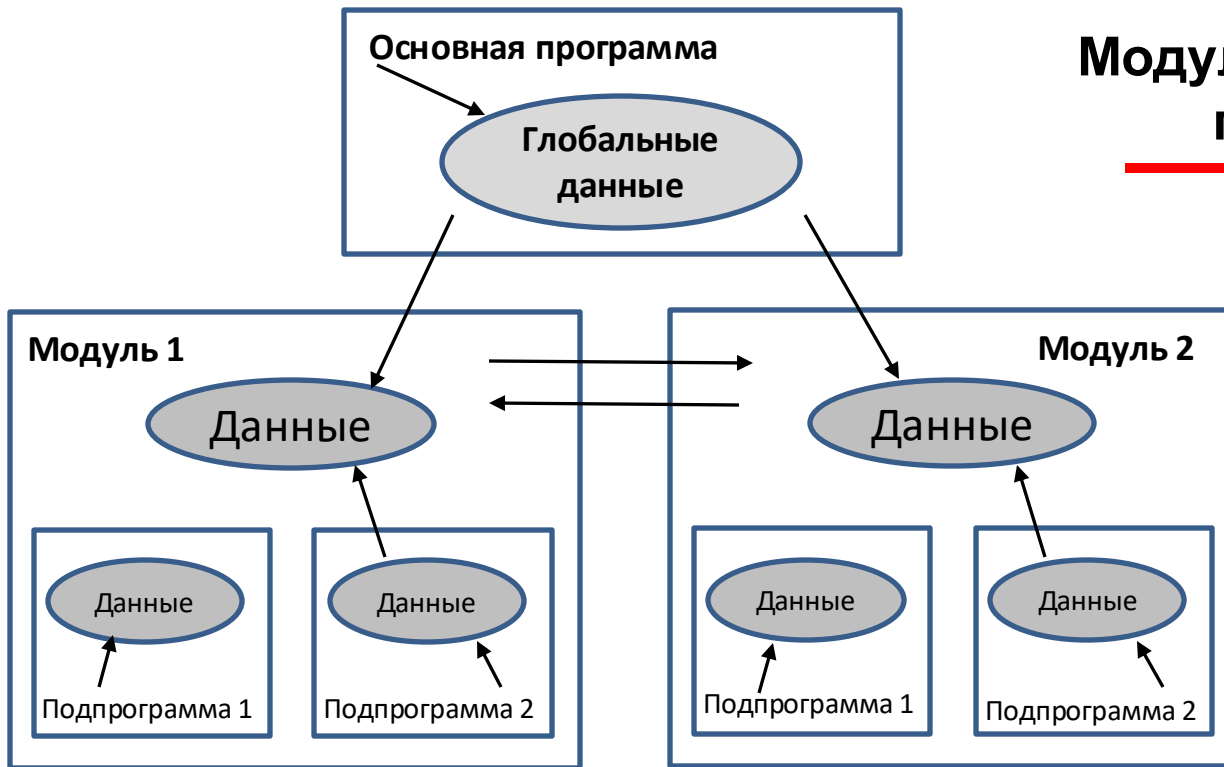


Модульная структура программы



- + Улучшение защиты данных - обращение к внутренним переменным и подпрограммам модуля только через интерфейс, прямой доступ запрещен.
- При увеличении количества модулей растет сложность межмодульных интерфейсов, трудно предусмотреть взаимовлияние модулей друг на друга.

Модульная структура программы



Модульный подход и сокрытие данных

- Реши, какие требуются модули.
- Разбей программу так, чтобы скрыть данные в модулях.

Инкапсуляция в процедурном подходе

```
1  unit StackUnit;
2  interface
3
4  type
5      TStack = Pointer; // Указатель на структуру стека, чтобы скрыть её реализацию
6
7  // Процедуры и функции для работы со стеком
8  procedure Push(S: TStack; Value: Integer);
9  function Pop(S: TStack): Integer;
10 function IsEmpty(S: TStack): Boolean;
11 function Top(S: TStack): Integer;
12 procedure FreeStack(var S: TStack);
13
14 implementation
15
16 type
17     PNode = ^TNode;
18     TNode = record
19         Data: Integer;
20         Next: PNode;
21     end;
22
23 // Внутренняя реализация структуры стека
24 type
25     TStackImpl = record
26         Head: PNode;
27     end;
28
29 // Функция создания нового стека
30 function NewStack(): TStack;
31 begin
32     New(Result);
33     with TStackImpl(Result)^ do
34         Head := nil;
35 end;
```