Информатика 3

Курс лекций

Игорь Павлович Соловьев / Доцент каф. инф. / i.soloviev@spbu.ru

Отчет за семестр — кому как повезет.

1. Зачет.
2. Экзамен

На лекциях — часть вопросов, избранная, с учетом длительности курса.
На практике — другая и в ином аспекте (вопросы практической реализации).

Часть 1. Алгоритмы и структуры данных (Представление данных)

Введение

Данная часть курса — в некотором смысле продолжение курса информатики, но не буквальное. В информатике в целом то же содержание — структуры данных и алгоритмы их обработки, а также вопросы их реализации.

Интересуемся

Современным взглядом на эту область, историей становления, а также существующими проблемами.

Информация и данные

Формально различают понятия **«информация» и «данные».**
При этом нам, как правило, наиболее интересна позиция специалистов в области кибернетики и информатики.

Информация

Н. Винер в книге «Кибернетика, или управление и связь в животном и машине; или Кибернетика и общество» пишет:

«Информация — это не материя и не энергия, информация — это информация».
Но основное определение информации, которое он дал в нескольких своих книгах, следующее.

Информация — это **обозначение** содержания, полученное нами из внешнего мира в процессе приспосабливания к нему нас и наших чувств.
(Объективность и отображение в запись, т.е. наличие описания.)

Также Винер и его последователи под информацией часто понимают то, что может вызвать определенные последствия (в результате нашего **осознания** этого содержания) (т.е. информация не нейтральна.)

Наш комментарий

Код, восприятие, интерпретация.

По сути оба пункта говорят об одном и том же разными словами и подразумевают объективность информации и наличие некоторого «кода» для обозначения ее содержания (отображение в запись — наличие описания) и метода интерпретации кода.

Данные

В литературе можно встретить различные определения понятия «данные».

Пример («определения») (безымянный и не бесспорный).

"Данные в компьютере представляются в виде кода, который состоит из единиц и нулей в разной последовательности. Код — набор условных обозначений для представления информации. Кодирование — процесс представления информации в виде кода".

Кодирование в некотором смысле теряет часть информации
(различные способы ее интерпретации.)

Замечание

1. Насколько формально различаются понятия информации и данных.

Здесь лучше говорить «данные», а не «информация», но в любом случае это различие (между понятиями И. и Д.) неформально;
инф активна, данные пассивна;
В то же время нам часто хочется знать, какую инф представляют данные.

2. Уровень абстракции и представление данных.

Нас это «определение» не устраивает в силу своей ограниченности, а именно, здесь выбран фиксированный уровень абстракции, не учитывающий всех деталей процесса восприятия инф, проектирования решения целевой задачи и реализации проекта; это станет ясно постепенно из других частей курса.
Мы всегда будем соотносить представление данных с определенным уровнем абстракции.

Уточним определение.

АиСД или **ПД (Data Representation) как дисциплина** изучает методы компьютерного представления данных или описания данных на языке, понятном операционной среде и/или специалисту на подходящем уровне абстракции в контексте решения определенной задачи или класса задач, а также вопросы реализации этих методов в виде компьютерных алгоритмов или систем.

Интуиция и эрудиция?

Задача ПД

Всегда включает в себя по крайней мере два аспекта/две стороны проблемы:

1) математический
(формально-теоретический) и
2) компьютерный
(реализационный, технологический)

М. аспект отвечает за выбор метода формализации описания данных и поиска точного (математического) решения.
К. — за выбор подходящего компьютерного языка описания на выбранном уровне абстракции и компьютерную реализацию выбранного метода представления.

Таким образом, ПД — ветвь компьютерной **науки** как раздела **прикладной математики**.
В то же время ПД — часть компьютерной **технологии**.

Граница между подходами

Оба аспекта обычно рассматриваются вместе, их трудно формально разделить.

Средства описания данных и алгоритмов

Для описания данных и алгоритмов на лекциях будем использовать различные языки, удобные в определенном контексте, от машинного до высокоуровневого языка спецификаций, включая псевдокод.

Список литературы (по 1 части, далеко не исчерпывающий)

1. Кнут. Искусство программирования. т. 1-3.
2. Кормен и др. Алгоритмы. Построение и анализ.
3. Ахо, Хопкрофт, Ульман. Структуры данных и алгоритмы.

Содержание.

4. Ахо, Ульман. "Компиляторы. Принципы, технологии, инструментарий", 2008
5. Учебники по C++, Java, C#, Паскалю и тд.
6. Гради Буч. Объектно-ор. анализ и проектирование с примерами на C++. 1998.
7. Дискретный анализ
   (Книга или лекции Романовского)
8. В.С. Зубов. Справочник программиста, 1999.
9. Вирт — два издания
10. Ускова О.Ф. и др. Пр-е алгоритмов обработки данных. СПб, БХВ, 2003.

Формирования методов представления данных в контексте истории развития методов программирования

История пр-я как науки и технологии — в некот см это и история созд методов ПД. Кратко — это история постепенного повышения уровня абстракции Д, усложнения стр-р Д, накопления неструктурир Д и методов выявл из них новых Д и знаний.

В прошлом часто с понятием иск-ва пр-я связывали понятие стиля пр-я.
Стиль пр-м на некот языке определяют лежащ в его осн содержат идеи.
Это не способ внешнего оформл пр-м, а стройн сист приемов и методов, предн для

- реш нек класса задач из опред предм обл или
- задач, объед по принципу сходства методов реш.

Сначала этот стиль в основном диктовался след обстоят.

- Представлениями первых компьют теоретиков,

др словами, состоянием теории алгоритмов (или, как тогда гов, алгорифмов) и информатики (в амер терминол — computer science).
В частн, недостаточной изученностью/разработанностью алгоритмической стороны вычислительного процесса —
текущее состояние теории.

- Архитектурой первых выч машин (40-е г) и состоянием техники и комп технологии в целом.

Хотя комп предн для помощи людям, возник при их использ трудн столь велики, что язык опис и реш проблемы разраб применит к инженерн реш, залож в констр ЭВМ.
Первые маш облад т.наз **арх фон Неймана** (один из первых и ведущ разработчиков совр ЭВМ).

Машина фон Неймана —

это модель компьютера, облад след особенностями:

Замечание

Программы как данные.
Фактически, пр-мы в арх фН сразу тракт как данные.

Данные — числовые коды
1я форма предст данных — числовые коды.

Бинарные, но
Почти сразу стали исп 8 и 16 разр коды
"Первый" стиль пр-я — императивный

(кавычки — поск никто о стиле тогда еще не говорил) — («приказной» - пр-мы сост из прямых инструкций процессору),

осн представители — всевозм маш языки, ассемблеры.

След этап — **поиск усовершенствованных языков** —
рез-т быстрого роста объемов пр-м, стимулируемый:
а) военными, научными, коммерч разр,
б) попытками решать задачи ИИ
(распозн речи, искусст и естест яз, перевод, криптография, управл сложн сист).

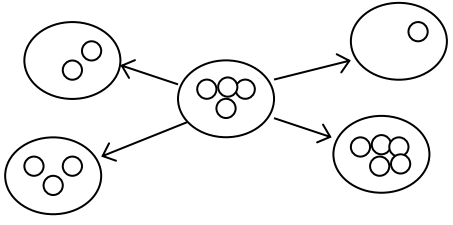
Первое усоверш, остающ в рамках того же стиля пр-я **—** Фортран (50е годы - Бэкус):
укрупненные инструкции, процедуры —те же укрупн инстр;
(попул и сейчас, созданы обширные библ станд проц.)
Возникло уточнение названия первого стиля пр-я —

процедурный или **процедурно-ориентированный стиль.**
(Фортран – чаще в военн и научн сферах, Кобол — в коммерч разр)
Данные по-прежн хр как числ коды – скаляры, векторы, матрицы, однако в яз пр-я появ **более высокий уровень абстр** – симв данные (строки).

След этап развития **—** требования **надежности и эффективности проектирования**, возник при разр больших прогр комплексов.
Пришло время системных преобразований — от случайных изобретений к разработке методов управления данными.

Акцент в прогр сместился (60е-70е).

- в сторону усоверш организации данных (научные исследования)
- и развития концепции модульности



Двоякая цель модульности

1. структуризация прогр комплекса в целом и данных — в частности,
2. «упрятывание» данных и процедур внутри модуля (фактически пришли к идее инкапсуляции).

Д. Парнас
Впервые удачно сформулир св-ва модуля: «Для напис одного мод д.б. дост миним знаний о тексте друг мод».
Это относ к ∀ проц или ф-и как ниж, так и верх ур иерарх (реализ).
Т.о. Парнас первым четко выразил идею сокрытия (инкапсуляции) инф в пр-и.

Глобальные переменные
Однако в 70е г проц и ф-и не обесп надежн сокрытия инф., поск подводило влияние глоб пер — их повед трудно предск в больш пр-мах (неразвитые яз пр-я).
Это м считать началом нового периода.

Модуль
Новая синт конст, не подверж влиян глоб пер,
модуль, появ в новых яз пр-я.
Использ наравне с проц и ф.
Это прототип совр понятий класса и объекта.

Структурное программирование

Возникли нов языки и новое усоверш в рамках той же идеологии, кот назвали **структурное пр-е**.
Осн представители **—** Алгол-60 (затем 68), PL1, Паскаль, Модула-2, Си и др.
М гов об огромн многообразии яз, возн в эти годы.

Итак, к нач 70х — пр-е проекты стало слишк сложны для проектир, кодир, отладки **в приемл сроки**. Колич и размер пр-м сделали процесс разр неуправл.
В итоге, этот этап породил новое усоверш **— структурное программирование**

Струкурная методологиея
Ведущ спец теории и практики пр-я —
Дейкстра, Вирт, Хоар, и др. разр строг прав ведения прогр проектов и назв их **структурной методологией.**

Осн идеи
(Дейкстра, неординарн личн) :

- пр-ма — иерархич совок абстрактн уровней

(глубокая идея);

- «оператор goto нужно считать вредным»

И все это для того, чтобы выполн основное треб **—**
контроль правильности данных на стадии созд и на стадии вып.
Отметим, что разр научно обоснованных методов всегда начин с навед элем порядка или систематизации.

Цели структ пр

1. Дисциплина

обесп дисц пр-я «кот пр-ст навяз себе сам» (Дейкстра);

2. Читабельность

улучш читабельность пр-м

Структурное пр-е предпол, в частн, особ стиль оформл текстов пр-м — ступеньками, этот стиль сохр попул и сейчас (маш яз, ассембл, фортран навязывали оформл в столбик)

Более точно, осн треб к текстам пр-м:

- избег языковых констр с неочев семантикой;
  напр, в яз Си допустима пр-ма, рез работы кот противор очевидному смыслу:
  ```
  #include <stdio.h>
  main()    {if (3<2<1) printf("чепуха");
             else printf("а  паскаль это не допускает");}
  ```
  Упр. проверить это в C и C#
- стремл к локализации действий управл констр и используемых стр-р данных;
- разраб пр-му: ее текст м читать от нач до конца без упр переходов на др страницу.

3. Эффективность разработки
повыш эфф процесса разр пр-мы (и самой пр-мы): структурир пр-му, разбивая ее на мод — легко нах и корр ош, независ переделки ∀ мод.

4. Надежность программ
повыш надежн пр-м: если легко поддается сквозн тестир, а это — при хор структурир (разб на мод и др стр-е единицы) и читабельности;

5. Время и стоимость
уменьш время и стоим разраб (!) — если повыш произ труда пр-ста; это обесп-ся соблюд правил стр пр-я.

Осн принципы стр-й методологии:

1. Абстракция

пр **абстракции** — предст решения без нек деталей; позв предст весь пр-й проект в виде неск ур абстр: верхн — весь проект (крупн детали), послед ур — постепенн детализация;

на этом принц осн разл **стратегии проектир**, напр, восходящ и нисходящ.

2. Формализация

принц **формализации** — строгий методич подход:

- база для превр процесса пр-я из импровиз в строг **инженерную** дисципл;
- дает осн для доказ правильн пр-м, т.е. пр-мы изуч как матем объекты (данные);
- иногда этому принц люди сопротивл, но он позв ускор принятие реш и избег ош.

3. Фрагментация

принц **«разделяй и властвуй»** — разб трудн пробл на много мелк фрагм (мод), простых по упр, допуск независ отл и тестир; дает возм независ разр отд частей проекта, не забот об огр кол деталей, охват всею систему;

модульное пр-е: один из осн методов структ-го пр-я.

4. Иерархия

принц **иерарх упр** (связ с предыд): важен не только факт разб, но и его стр-ра; вообще иерарх стр помог эфф упр; в частн, в пр-и иерарх стр помог достиг целей стр-го пр-я.

Резюме. Основные идеи стр пр:
надежность, контроль правильности данных, эффективность разработки и эксплуатации ПО.

Стр пр — **крик души** обеспокоенных специалистов, однако **рядовые пр-сты** не любят следовать советам.
Стр пр-е помогло на опр **этапе**. Однако **одна идея не м** реш всех пробл в экон и прогр проектировании — объемы и сложн ПО только растут.
В то же время стр пр, **как таблица умн**, изобрели и используют.

След этап — ООП.

Абстракция данных, Симула-67, CLU

В **русле данного напр** в посл трети 20го века наметился нов подход **—** ООП.
Из систематизации (стр пр) выросла новая технология.
Симула-67. CLU
Возникло новое понятие — абстракция данных; первые языки — Симула-67 – группа в Норвегии и CLU (автор – амер матем и пр-ст Барбара Лискофф, 70е).

Основные идеи ООП

Предпол более решительный шаг — связывание данных и процедур, их обр, в единое целое — объекты (инкапсуляция).
Объекты м взаимодействовать как активн агенты.
В отличие от других расширений языка, имеющих в основном технологический характер (иногда - косметический), — строковых типов, модулей, оверлеев и т.д., введение объектных типов — (потенциально) достаточно радикальное изменение подхода к программированию.
От во многом случайного и механич объед проц и данн в модули акцент смещ в сторону систематич и смыслового объед программных объектов.

Программирование в терминах предметной области

Развитие идеи пр-я **в терминах предметной области** — смысловая структуризация. Не отмена традиционного подхода, а согласованная совок. новых принципов и понятий, позв существенно повыс произв. труда пр-стов, сдел пр-мы более комп., легко расш. И более понятн
Использование (на раннем этапе) - дело пр-стов (формально не навяз, но технологически обусловл).
Непривычно, но изучение даст выигр. в дальн. работе.

80-е, 90-е, 2000е

ООП активно развивается в 80-е, 90-е, 2000е в виде большого числа новых яз пр-я.
В 80е — Smalltalk (заимствовал идеи из Simula 67 — 60е).
Середина 80х — Object Pascal фирмы Apple, затем BP.
Затем Си++ — ОО расш. Си, Java-технол, C#, dotNet и т.д.
На начальн. этапе ООП — в осн. исслед. интерес., большие накл. расх.
Сейчас — высоко эфф. компил. => м. исп. для промышл. проектир.
(Пример — ОО пакет Turbo Vision, на основе кот. разр. интегр. среда TP.)
Итак, еще раз подч, что преимущ ООП сказыв только на прогр дост большого объема — 1000 и более строк исх текста, на простых прогр имеет скорее методич (учебное) знач.
Большие пр-мы треб структурирования — преобр их в пр-е комплексы, здесь и помог ООП.

Главное, на **совр этапе** от созд разрозненного набора новых яз перешли к созд **новых технологий**, причем целого спектра, напр, ООтехн, Java, dotNet, КомпОриент техн, АспектноОр Техн и др.
При этом в процессе эвол стилей и методов пр-я развивались и **методы ПД**.

Подч, что ∃ и др усоверш стилей и методов пр-я — другие парадигмы, напрЛП и ФП, интегрированное напр ЛФП, но этот вопр вых за пред данного курса.

Сложность, присущая программному обеспечению

Брукс, Эйнштейн
История разв методов пр-я и, в частн, ООП — это и история преод трудн, связ со сложн пр-го обесп. В основе многих событий и закономерностей этой истории было то, что. пр-ным проектам присуща сл особого рода.

Изв спец в обл пр-го проект Брукс: «Эйнштей утвержд, что должны существов прост объясн прир процессов, т.к. Бог не действ по капризу или произволу. У пр-ста нет такого утеш: сложность, с кот он д справиться, лежит в самой природе системы.» (которую, мы добавим, создали люди)

Почему пр-му обеспечению присуща сложность

(Речь идет о больш промышл проектах, а не о разработках одиночек.
Гл черта пром пр — **один разр-чик в принц** не м охв все аспекты такой системы)

1. Предм область
сложна реальн предм обл, из кот исх заказ на разработку;
2. Управление
трудн упр процесса разр
(много разработчиков, функция сложности управления проектов — примерно факториал числа участников, если только пары — примерно квадрат);
3. Гибкость
необх обесп достат гибкость пр-мы
(много параметров и противоречивость требований);
4. Эволюция
необх обесп возм эвол со временем;
напр, по нормативам военн корабли и подлодки д допуск модернизацию в проц экспл до 30% технол оснастки и вооруж.
5. Описание систем
неудовлетворительными способами описания структуры и поведения больших дискретных систем.

С 1 – 4 мы ничего не м поделать.
5 — это то, чем мы можем в какой-то мере управлять, по кр мере изучать и совершенствовать.

Декомпозиция

Внес порядка в сложн систему
(наша задача) — **декомпозиция** (уже отмечали) – необходимая часть процесса проектирования.

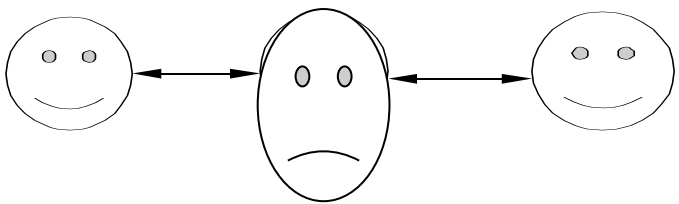
Как осуществить:

- **Алгоритмическая декомпозиция**

— разб на процедуры и мод — предл структ пр-е «сверху вниз»
— Концентрация вним на порядке событий.
(характерно для ранних этапов развития компьютерной науки и технологии, в частн, стр-е пр-е)

- **ОО декомпозиция**

— декомп по принц принадл к разл абстракциям данных предм области (этим и б заниматься)
— конц вним на агентах, действ в предм области, кот явл либо объектами, либо субъектами действий.



С чего лучше начинать и почему
Важны оба спос декомп, но лучше начин проектир с ОО декомп. Ее преим:

- ОО дек уменьш размер пр-х систем за счет повторного использ общ мех и пр-х комп — сущест экон выразит ср-в;
- ОО сист более гибки и проще эвол со временем, т.к. их схемы базир на устойч формах; большие сист развив из маленьких, уже проверенн;
- проще разобр в сложн пр-й сист. (проще анализ), поскольку улучш структура прогр проекта, напр, иерарх стр.

Зам. Дополнительные источники сложности

Резюмируя, м дополнительно заметить, что сложность решения задачи имеет две стороны или два источника:
1) научно-практическая проблема, кот следует решить, напр, математическая или другая точная естественно-научная задача;
2) существующая (применяемая) технология реализации решения.

Как правило очень трудно, иногда невозм провести строгую границу между этими сторонами проблемы. Увидим это на примерах.

Методы оценки сложности алгоритмов

Пусть $n \in \mathbb{N}$ — наиболее существенный **параметр** некоторой задачи или код нескольких таких параметров задачи, напр, номер кортежа этих парам в некоторой нумерации.
И пусть требуется **оценить время** (дискретное) работы алгоритма, решающего эту задачу или **объем памяти** (в некоторых единицах памяти), требуемых для решения этой задачи при неограниченном увеличении значения параметра.

Единицы измерения могут быть абстрактным или конкретными.

При этом время или память оцениваются как некоторая **функция *f*** данного парам ***n*.**
Ее иногда наз **мерой** или **функцией сложности** задачи.

Опр. Пусть даны неотрицательные функции ***f*** — ф-я **сложности** и g, наз. **оценкой** сложности.

Для ф-й, кот м иметь отриц зн, обычно для оценок использ их **абс** величины.

Введем три **метода** оценки сложности ***f*** посредством свойств:

$f(n) = O(g(n)) \Leftrightarrow$

$\exists c > 0 \ \exists k \ \forall n \geq k \ f(n) \leq cg(n)$ — g —оценка f сверху

$f(n) = \Omega\big(g(n)\big) \Leftrightarrow$

$\exists c > 0 \ \exists k \ \forall n \geq k \ f(n) \geq cg(n)$ — снизу

$f(n) = \Theta\big(g(n)\big) \Leftrightarrow$

$\exists a, b > 0 \ \ \exists k \ \ \forall n \geq k \ \ a \cdot g(n) \leq f(n) \leq b \cdot g(n)$ — «точная»

Асимптотические оценки

Эти оценки часто наз **асимптотическими**.

Упр. Выполняются три свойства:

- $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$, но не наоборот (аналогично для Ω).
- $f(n) = \Theta\big(g(n)\big) \Leftrightarrow f(n) = O\big(g(n)\big)$ & $f(n) = \Omega\left(g(n)\right)$

Чаще используют оценку $O(g(n))$, т к

обычно хотят иметь гарантию **ограничения** скор роста ф-и сложности, но поиск хорошей оценки сверху м б трудной задачей

Примеры.
Гамильтонов цикл графа, выполнимость булевской формулы.

Но на практике также часто интересуются и оценками скор роста **снизу**
(для не очень хороших задач)
и точной
(не может быть хуже или лучше).

Обычно оценку **сверху** получают
для всех возможных значений параметра, даже самых "плохих", поэтому часто эти оценки наз "оценки **в худшем** случае".

"**в среднем**", "**в лучшем сл**"
Однако также для самых интересных или часто встречающихся значений параметра находят оценки "**в среднем**" (быстрая сорт.), а иногда, редко, и "**в лучшем сл**" (снизу).
Это необходимо всегда оговаривать.

Пример. Сортировки.

1. Простые (вставки, пузырек и т.п.).
Сложность ("время" вычисления) $f(n) = O(n^2)$.
Это оценка времени сверху, причем в худшем случае.
Оценка памяти для хранения списка: $O(n)$, где n длина списка.

2. Быстрая.
Сложность ("время" вычисления) $f(n) = O(n \log n)$.
Это оценка времени "**в среднем**".
Эта оценка важна, т к этот средний случай чаще всего и встречается.
Но **в худшем** все равно $f(n) = O(n^2)$.

3. Графы.
Оценка памяти для хр графа для разл способов предст:
$O(n^2)$, $O(n+m)$, где n и m — кол узлов и дуг графа.