

Федеральное государственное автономное образовательное учреждение высшего образования  
Национальный исследовательский университет «Высшая школа экономики»

**Незнанов А.А.**

**Базовый тезаурус по дисциплинам цикла  
«Программирование» в бакалавриате**

Используемая терминологическая система с английскими эквивалентами

© 2011-2018, Незнанов Алексей Андреевич

Версия 1.00.03

---

Автор выражает благодарность Дегтярёву К.Ю., Кохову В.А., Кутепову В.П.,  
Меньшиковой К.Г. за плодотворные дискуссии и ценные терминологические замечания.

# Содержание

|  |           |
|--|-----------|
| <b>Раздел 1. Программный продукт как объект исследования.....</b>          | <b>3</b>  |
| 1.1. Процесс абстрагирования и уровни абстракции.....                      | 3         |
| 1.2. Алгоритм и программа .....  | 4         |
| 1.3. Программная архитектура .....   | 5         |
| 1.4. Основные характеристики программных продуктов .....                   | 5         |
| 1.5. Процесс создания программного обеспечения.....                        | 6         |
| <b>Раздел 2. Компьютер как исполнитель алгоритмов.....</b>                 | <b>8</b>  |
| 2.1. Архитектура компьютера.....   | 8         |
| 2.2. Основные принципы повышения эффективности работы компьютеров ..       | 9         |
| <b>Раздел 3. Языки программирования .....</b>                              | <b>10</b> |
| 3.1. Парадигмы программирования .....                                      | 10        |
| <b>Раздел 4. Синтаксис и семантика языка программирования .....</b>        | <b>12</b> |
| 4.1. Базовые лексические единицы исходного кода.....                       | 13        |
| 4.2. Базовые синтаксические единицы исходного кода.....                    | 14        |
| 4.3. Переменные и константы.....   | 14        |
| 4.4. Реализация языка программирования (трансляторы) .....                 | 15        |
| 4.5. Модуляризация.....  | 16        |
| <b>Раздел 5. Подпрограммы.....</b>   | <b>17</b> |
| <b>Раздел 6. Данные и их структуры.....</b>                                | <b>18</b> |
| 6.1. Индексы и ссылки.....   | 20        |
| 6.2. Абстрактные типы данных.....  | 21        |
| 6.3. Структуры данных как механизм реализации АДД .....                    | 23        |
| <b>Раздел 7. Объектно-ориентированное проектирование и конструирование</b> | <b>25</b> |
| 7.1. Концепции объектно-ориентированного подхода .....                     | 25        |
| <b>Раздел 8. Информационный поиск и сортировка .....</b>                   | <b>26</b> |
| <b>Раздел 9. Вычислительная сложность.....</b>                             | <b>27</b> |
| <b>Раздел 10. Рекомендуемые источники информации .....</b>                 | <b>29</b> |
| 10.1. Основные публикации .....  | 29        |
| 10.2. Некоторые Интернет-источники.....                                    | 31        |
| <b>Раздел 11. Список общепринятых аббревиатур .....</b>                    | <b>32</b> |
| <b>Раздел 12. Терминологический указатель.....</b>                         | <b>33</b> |

# Раздел 1. Программный продукт как объект исследования

## 1.1. Процесс абстрагирования и уровни абстракции

**Система** [*system*] – объединение множества элементов некоторыми связями, у которого появляются свойства, отсутствующие у любого элемента по отдельности. Такие свойства называют системными свойствами, а их появление – проявлением **эмерджентности** [*system emergentness*] (от англ. *emergence* – возникновение, появление, непредвиденный случай). Эмерджентность не позволяет свести свойства системы к сумме свойств её элементов. Системы всегда образуют иерархию, то есть можно считать, что у любой системы есть *надсистема* и *подсистемы*.

**Подсистема** [*subsystem*] – система, рассматриваемая как элемент системы более высокого уровня.

**Надсистема** [стандартного термина нет, иногда используется *supersystem*] – система, для которой рассматриваемая система является элементом.

**Состав** [*composition*] системы – совокупность элементов, образующих систему, то есть их перечисление.

**Структура** [*structure*] системы – совокупность связей между элементами системы, то есть способ объединения элементов, обеспечивающий эмерджентность.

**Абстракция** [*abstraction*] – 1. Разграничение внешних (существенных с точки зрения надсистем) свойств системы и внутренних деталей её строения и функционирования. 2. Принцип моделирования, заключающийся в игнорировании аспектов проблемы, не оказывающих существенного влияния на её решение. Хорошей абстракцией считается та, которая выделяет свойства системы, не зависящие от реализации подсистем и действительно существенные для рассмотрения и использования в данном контексте, опуская все остальные.

**Иерархия абстрактных представлений (иерархия абстракций)** [*abstraction hierarchy*] – относительное упорядочение абстракций по структурам классов, объектов или процессов.

**Уровень абстракции** [*level of abstraction*] – уровень в иерархии абстракций.

**Интерфейс** [*interface*] – одно из базовых понятий теории систем. В широком смысле, интерфейс – это формальное или неформальное определение связи в тройке «сущность1 – связь – сущность2» [*entity1 – relation – entity2*].

Два более узких определения понятия «интерфейс» – 1. Часть системы, открытая (доступная) для других систем, внешнее восприятие системы. 2. Система связей с унифицированными сигналами и/или аппаратурой, предназначенная для обмена материей/информацией между техническими устройствами и/или людьми.

## 1.2. Алгоритм и программа

*Информация, данные и алгоритм* – базовые понятия, которые невозможно описать в полном объёме через другие понятия (проблема порочного круга). **Информация – не материя и не энергия!** (Н. Винер, 1948 г.).

**Алгоритм** [*algorithm*] – чёткое конечное описание последовательности элементарных *действий* исполнителя над исходными данными для достижения некоторой цели.

**Данные** [*data*] – любая информация, представленная в форме, пригодной для хранения, передачи и обработки **средствами вычислительной техники (компьютерами)**. Будем считать, что исполнителем алгоритмов обработки данных является **компьютер** (см. раздел 2). Подробнее о данных – в разделе 6.

**Программа** [*program*] (согласно ГОСТ 19781-90 «Обеспечение систем обработки информации программное. Термины и определения») – *данные*, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма. Другое, более общее и понятное определение – *формальная запись алгоритма и обрабатываемых им данных на языке программирования для некоторого класса исполнителей*.

Текст программы называется **кодом** [*code*]. Подробнее о коде и его преобразовании – в разделе 4.4.

**Информационная система** [*information system*] – система, обеспечивающая сбор, хранение, обработку и доступ пользователей к информации.

**Программное обеспечение (ПО)** [*software*] – общее понятие, описывающее программы для компьютеров в отличие от их аппаратуры (которая стала обозначаться термином *hardware*).

**Программный продукт (ПП)** [*software product*] – 1. *Готовая (работоспособная) программа*, предоставляемая потребителю совместно со средствами *развёртывания*, *необходимой документацией*, *лицензией* и *гарантиями*. 2. Самостоятельное, отчуждаемое произведение, представляющее собой публикацию текста программы или программ на языке программирования или в виде исполняемого кода (согласно ГОСТ 7.83-2001 «Электронные издания. Основные виды и выходные сведения»).

**Программирование** [*programming*] – область инженерной деятельности, связанная с созданием программ для автоматизированных технических устройств, то есть вся совокупность процессов, связанных с разработкой ПО.

**Конструирование ПО** [*software design*] – процессы выбора структур данных и действий, кодирования [*coding*], оптимизации [*optimization*], отладки [*debugging*] и тестирования [*testing*] программ.

**Инфраструктура** [*infrastructure, framework*] – комплекс взаимосвязанных обслуживающих структур, составляющих и/или обеспечивающих основу для решения некоторого класса проблем.

### 1.3. Программная архитектура

**Архитектура программной системы** [*software architecture*] – совокупность существенных решений, определяющих:

- 1) общую организацию системы;
- 2) выбор уровней абстракции и структурных элементов системы, а также интерфейсов между ними;
- 3) поведение структурных элементов в процессе взаимодействия с другими элементами;
- 4) архитектурный стиль, направляющий и определяющий организацию системы, процесс абстрагирования и реализации.

Следует понимать, что главное свойство архитектуры сложной системы – многоуровневость. На самом верхнем уровне архитектуры располагается **концепция** (от лат. *conceptio* – понимание), то есть определённая точка зрения на систему, руководящая идея при её исследовании или создании. Ниже обычно выделяют **логическую** [*logical*] и **физическую** [*physical*] архитектуру. Построение архитектуры всегда является процессом *специфицирования*.

**Спецификация** [*specification*] – детализированное (относительно текущего уровня абстракции) определение чего-либо, включающее классификацию и перечень *специфических* особенностей, важных параметров и их значений.

**Специфика** [*specificity, peculiarity, specific characters*] – совокупность специфических особенностей чего-либо.

### 1.4. Основные характеристики программных продуктов

**Назначение** [*purpose*] – цель создания с точки зрения пользователя, предполагаемая роль и функция ПП. ПП должен отвечать своему назначению, которое должно быть сформулировано как можно точнее.

**Целевая аудитория** [*target audience*] – предполагаемые пользователи ПП, с учётом чьих потребностей формулируется назначение.

**Условия эксплуатации** [*operating conditions*] – спецификация параметров внешней среды, обеспечивающих нормальное функционирование ПП (необходимые устройства и их параметры, программное окружение и т.п.).

**Объём** [*size*] – 1. Общая характеристика размера готового ПП, измеряемого в количестве информации. 2. Характеристика функциональной насыщенности ПП.

**Сложность** [*complexity*] – 1. Качественная характеристика, выражающая трудность ПП для понимания и обучения. 2. Временная и емкостная сложность используемых реализаций алгоритмов.

**Эффективность** [*efficiency*] – способность наиболее полно использовать предоставляемые ресурсы. Под ресурсами понимаются: процессорное время, необходимый объём оперативной памяти и внешней памяти, пропускная способность каналов связи. Не путайте эффективность с **производительностью** [*performance*] – способностью выполнять как можно больше задач за как можно меньшее время. Например, из двух программ более эффективной является та, которая при *равном потреблении ресурсов* обеспечивает лучшую производительность или при *равной производительности* потребляет меньше ресурсов. Если в контексте ресурсы фиксированы, то эти термины могут употребляться как синонимы.

**Корректность** [*correctness*] – способность выполнять задачи так, как это определено в спецификации ПП.

**Устойчивость** [*robustness*] – способность адекватно реагировать на возникновение аварийных ситуаций и восстанавливаться после них.

**Совместимость** [*compatibility*] – способность совместной работы и обмена информацией с как можно большим числом различных устройств и других ПП.

**Расширяемость** [*expandability* или *extendibility*] – 1. Способность лёгкой адаптации к изменениям спецификации. 2. Возможность наращивания функциональности без изменения исходного ПП. Обычно реализуется посредством интерфейсов прикладного программирования (*API*), встроенных макросредств, языков программирования и т.п.

**Открытость** [*openness*] – доступность подробных спецификаций, обеспечивающих расширяемость и облегчающих достижение совместимости.

**Переносимость** [*portability*] – легкость переноса ПП на другие платформы (архитектуры компьютеров, операционные системы).

**Универсальность** [*versatility*] – пригодность ПП к использованию при решении как можно большего класса задач как можно большим числом пользователей.

## 1.5. Процесс создания программного обеспечения

**Платформа** [*platform*] – наиболее общее описание программно-аппаратной (в том числе сетевой) среды, на которой разрабатывается и/или развёртывается прикладное ПО. Обязательно включает архитектуру компьютера, тип ОС и средства доступа к данным (СУБД). Часто употребляют термины **платформа разработки** [*development platform*] и **платформа развёртывания** [*deployment platform*]: средства

разработки ПО и необходимое программное окружение готового ПО соответственно.

**Лицензионное соглашение** [*license agreement*] – юридически оформленное соглашение между владельцем ПП с одной стороны и его пользователем с другой стороны. В нём изложены условия, на которых может использоваться данный ПП.

**Конечный пользователь** [*end-user, final consumer*] – физическое или юридическое лицо, которое использует готовый продукт производства, в данном случае – программный продукт. Его права изложены в **лицензионном соглашении с конечным пользователем** [*end-user license agreement – EULA*].

**Дистрибутив** [*distributive*] (инсталлятор) – специальный комплект ПО для автоматизированного развёртывания продукта на оборудовании пользователя.

**Жизненный цикл** [*life cycle*] программного обеспечения (ЖЦПО) – период разработки и эксплуатации программного обеспечения от решения о его создании до прекращения использования.

**Техническое задание (ТЗ) или спецификация требований к программному продукту** [*requirements specification*] – основной документ, определяющий требования и порядок создания, развития или модернизации ПП, в соответствии с которым проводят создание ПП и его приемку заказчиком.

**Документ** (в общем) [*document*] – артефакт, являющийся средством закрепления различным способом на материальном носителе сведений о фактах, событиях, явлениях объективной действительности и мыслительной деятельности человека.

**Реквизит документа** [*document entry*] – обязательный элемент оформления официального документа, установленный каким-либо соглашением, отсутствие которого влечет за собой потерю статуса (например, потерю юридической силы).

**Стандартизация** [*standardization*] – приведение результатов и приемов работы к единообразным нормам.

**Стандарт** [*standard*] – 1. Нормативно-технический документ по стандартизации, устанавливающий комплекс норм, правил, требований к объекту стандартизации и утвержденный компетентным органом. 2. Типовой образец, которому должно удовлетворять изделие по размерам, форме и качеству. 3. Эталонное значение некоторого параметра, согласованное несколькими заинтересованными сторонами.

**Рекомендация** [*recommendation*] содержит не требования, но предложения, которые не являются обязательными. Обычно эти предложения дополняют и развивают нормы, зафиксированные в стандарте.

**Руководство** [*handbook, manual, guide*] содержит указания по применению методов проектирования и конструирования в рамках некоторой методологии.

**Соглашение** [*convention*] – любая договорённость, не оформленная официально в виде стандарта.

**Соглашение о стиле** (руководство по стилю) [*style guide*] – документ, дополняющий грамматику языка программирования и описывающий, как нужно поступать, когда есть возможность выбора из многих синтаксически правильных вариантов при оформлении исходного кода.

**Соглашение об именовании** [*naming convention*] – договорённость об использовании некоторых правил присваивания имён (идентификаторов) объектам в зависимости от их типа, цели использования и отношений между ними.

## Раздел 2. Компьютер как исполнитель алгоритмов

### 2.1. Архитектура компьютера

**Архитектура компьютера** [*computer architecture*] – совокупность проектных решений, определяющих в общих чертах процесс обработки информации: логическую организацию, принцип кодирования информации, принципы выделения подсистем и их взаимодействия.

**Структура (организация) компьютера** [*computer structure*] – границы его подсистем и связи между ними на некотором уровне абстракции.

**Процессор** [*processor*] – устройство, способное выполнять некоторый набор команд или инструкций.

**Память** [*memory*] – устройство, способное принимать, хранить и избирательно выдавать данные.

**Оперативная память** (ОП) или оперативное запоминающее устройство (ОЗУ) [*main memory, main storage*] – память, связанная с процессором и хранящая команды и данные, непосредственно доступные процессору.

**Центральный процессор** (ЦП) [*central processor unit, CPU*] – программно-управляемое устройство, организующее процесс обработки информации и координирующее работу всех других устройств компьютера. Современный цифровой ЦП включает устройство управления, исполнительные устройства, регистры и кэш-память.

Остальные устройства компьютера называют **периферийными** [*peripherals*] или **устройствами ввода/вывода** [*input-output devices*], так как они предназначены для общения компьютера с внешней средой.

**Регистр процессора** [*register*] – сверхбыстрая память внутри процессора, предназначенная, прежде всего, для хранения промежуточных результатов вычисления (регистры общего назначения или регистры данных) или содержащая



данные, необходимые для работы процессора — смещения базовых таблиц, уровни доступа и т. д. (специальные регистры).

**Абстрактная машина** [*abstract machine*] – вычислительная машина (компьютер) как набор логических ресурсов процессора и памяти, способов их взаимодействия и быстрогодействия (то есть архитектура без конкретной реализации).

**Бит** [*bit*, от *Binary digIT*] – двоичная цифра, являющаяся по совместительству единицей количества информации. Обычно записывается символами «0» и «1».

**Байт** [*byte*] – последовательность бит фиксированной длины. В настоящее время наиболее употребимый размер байта – 8 бит. В большинстве практических применений количество информации измеряется именно в байтах и кратно ему.

**Слово** [*word*] – последовательность байт фиксированной длины. В настоящее время наиболее употребимый размер слова – 2 байта.

**Машинное слово** [*machine word*] – последовательность байт, длина которой равна размеру регистров общего назначения рассматриваемого процессора. Число бит в машинном слове называется **разрядностью процессора**.

## 2.2. Основные принципы повышения эффективности работы компьютеров

**Конвейеризация** [*pipelining, staging*] – разбиение решаемой задачи на несколько последовательных операций (стадий), которые выполняются на конвейере.

**Распараллеливание** [*parallelization*] – преобразование линейного алгоритма в параллельный (многопоточный и/или векторный) для эффективного использования многоядерных/многопроцессорных/многомашинных вычислительных систем.

**Спекулятивное выполнение** [*speculative execution*] – преждевременное выполнение команд, о которых достоверно не известно, действительно ли будет востребован результат их выполнения.

**Суперскалярность** [*superscalar*] – способность процессора выполнять параллельно несколько команд *последовательной* программы, для чего, соответственно, требуется несколько конвейеров.

**Кэш-память** (или просто *кэш*) [*cache*] – промежуточное (буферное) быстродействующее запоминающее устройство, служащее для нивелирования различий в параметрах работы (в первую очередь – скорости передачи данных и времени доступа) различных устройств, обменивающихся данными.

**Кэширование** [*caching*] – использование высокоскоростной (но и более дорогой) памяти для прозрачного промежуточного хранения наиболее часто

используемых информационных элементов некоторого хранилища с целью уменьшения времени доступа к данным.

**Выравнивание** [*alignment*] – расположение данных в памяти таким образом, чтобы отдельные информационные элементы имели адреса, кратные некоторому числу (как правило – степени двойки, в подавляющем большинстве случаев – разрядности регистров общего назначения центрального процессора) для ускорения доступа к элементам.

## Раздел 3. Языки программирования

**Язык программирования** (алгоритмический язык) [*programming language*] – формальная знаковая система, служащая для формального описания и реализации на компьютере алгоритмов обработки данных.

Запись команд процессора непосредственно в том виде, в котором они поступают на его вход, называется **машинным языком** (машинным кодом) [*machine language/code*] и осуществляется в двоичном (бинарном) коде.

**Язык ассемблера** [*assembler language*] – это символьная форма машинного языка с рядом возможностей, характерных для языка высокого уровня. С помощью языка ассемблера, как и с помощью машинного языка, программист получает доступ ко всем ресурсам компьютера.

**Язык программирования высокого уровня** (ЯПВУ) [*high-level programming language*] – 1. Язык программирования, понятия и структура которого удобны для восприятия человеком (согласно ГОСТ 19781-90). 2. Аппаратно-независимый язык программирования.

**Модель вычислений** [*model of computation*] – формальное абстрактное описание исполнителя алгоритма.

### 3.1. Парадигмы программирования

**Парадигма** [*paradigm*] (от греч. *paradeigma* – образец, пример для подражания) – схема, модель постановки проблем и их решения, методы исследования, господствующие в течение определенного исторического периода в научном или профессиональном обществе. Современные языки, как правило, являются *мультипарадигменными*.

Парадигмы программирования делят на два больших класса:

- 1) **императивные**, требующие явно описывать процесс изменения состояния исполнителя;
- 2) **декларативные**, требующие задать описание цели, а также понятия и факты, необходимые для её достижения.

### 3.1.1. Императивное программирование

**Императивное программирование** [*imperative programming*] описывает алгоритм в явном виде, позволяя задать последовательность изменений состояний исполнителя алгоритма во времени.

Императивное программирование напрямую следует из интуитивного понимания *пошагового описания алгоритма* и полностью соответствует принципам построения подавляющего большинства компьютеров (архитектуре фон Неймана).

**Структурное программирование** [*structured programming*] – методология и технология разработки программных средств, основанная на использовании трёх базовых конструкций (следование, ветвление и цикл) и принципах:

- 1) абстракции и инкапсуляции *процессов* в виде подпрограмм;
- 2) программирования «сверху-вниз» (нисходящего проектирования);
- 3) модульного программирования с иерархическим упорядочением связей между модулями.

### Объектно-ориентированное программирование

Объектно-ориентированная парадигма не отрицает структурную, а развивает её, по-новому реализуя принципы абстракции, инкапсуляции и декомпозиции (см. раздел 7.1).

В отличие от структурной, объектно-ориентированная парадигма (ОО-парадигма) программирования декларирует объектную декомпозицию программы, сводящуюся к выделению *объектов* и *связей* между ними.

В объектно-ориентированном программировании единицей декомпозиции становится *объект*, инкапсулирующий одновременно данные и процессы их обработки, а программа начинает управляться *событиями*.

Вычислительная модель чистой ОО-парадигмы описывается как обмен объектов сообщениями, которые также являются объектами. Каждый объект обладает состоянием, которое изменяется в результате обработки сообщений. Общее состояние вычислительной модели есть совокупность состояний всех её объектов.

### 3.1.2. Неимперативное программирование

**Декларативное программирование** [*declarative programming*](от лат. *Declaratio* – объявление) – программирование, не требующее описания процесса получения результата, а требующее лишь описать сам результат, хотя бы и достаточно формально, чтобы компьютер смог выполнить процесс его получения.

**Декларативный язык программирования** [*declarative language*](от лат. *Declaratio* – объявление) – язык программирования высокого уровня, описывающий

не императивный процесс получения результата, а строение исходных данных и необходимый вид результата.

**Функциональное программирование** [*functional programming*] – программирование в терминах композиции функций. Единственным действием функциональной программы является вызов функции.

**Язык функционального программирования** [*functional programming language*] – язык программирования, позволяющий задавать программу в виде совокупности определений функций.

**Логическое программирование** [*logical programming*] – программирование в терминах *фактов* и *правил вывода*, с использованием языка, основанного на формальных логических исчислениях.

**Язык логического программирования** [*functional programming language*] – язык программирования, позволяющий задавать программу в виде базы фактов (набора логических утверждений) и утверждения, нуждающегося в доказательстве.

## Раздел 4. Синтаксис и семантика языка программирования

**Семиотика** [*semiotics*] (от греч. *semeiotikón* – знак, признак) – наиболее общая наука (метанаука) о знаках и знаковых системах, основными из которых являются естественные и искусственные языки.

**Синтактика** [*syntactics*] (от греч. *syntaktikos* – строящий по порядку, приводящий в порядок) изучает внутренние свойства знаковых систем, не зависящие от интерпретации знаков.

**Семантика** [*semantics*] (от греч. *semantikos* – обозначающий) рассматривает отношение знаков к обозначаемому (содержание знаков, их смысл), то есть интерпретацию знаков, независимо от того, кто является интерпретатором – потребителем смысла.

**Прагматика** [*pragmatics*] (от греч. *prágma* – дело, действие) исследует связь знаков с их потребителями, то есть проблемы интерпретации знаков теми, кто их использует, их полезности и ценности для интерпретатора.

**Грамматика** [*grammar*] (от греческого *grammata* – письмена, писания) – система правил построения и описания естественного или искусственного языка, в том числе правил словообразования и построения словосочетаний (предложений). Грамматика делится на **морфологию** [*morphology*] (определяющую правила написания отдельных слов) и **синтаксис** [*syntax*] (определяющий способы

организации слов в словосочетания и предложения языка, а также типы предложений, их значения и условия использования в текстах).

**Умолчание** [*default*] – соглашение о характеристике языкового объекта или выполняемом действии при отсутствии их явного описания.

#### 4.1. Базовые лексические единицы исходного кода

**Алфавит** [*character set*] языка программирования – набор символов, используемый для построения текста программы.

**Лексема** [*lexeme*] – элементарная синтаксическая единица текста программы, то есть минимальная значимая последовательность символов алфавита. Описание лексем обычно даётся отдельно от синтаксического описания.

**Комментарий** [*comment*] – часть исходного кода, которая не анализируется (игнорируется) транслятором. Комментарии вводятся в исходный код с помощью специальных конструкций языка программирования.

**Идентификатор** [*identifier*] – имя сущности, обладающее следующими свойствами:

- 1) каждая сущность имеет только один идентификатор;
- 2) различные сущности не могут иметь совпадающие идентификаторы;
- 3) время жизни сущности совпадает со временем жизни идентификатора.

**Область видимости** или **область действия** [*scope, visibility scope*] идентификатора – участок текста программы, в котором этот идентификатор можно использовать. В различных областях видимости могут встречаться одинаковые идентификаторы, которые будут обозначать различные сущности. См. также *перегрузку подпрограмм*.

**Метка** [*label*] – необязательный специальный *идентификатор оператора* программы, служащий для указания места безусловной передачи управления оператором безусловного перехода *goto* (синтаксис может меняться).

**Ключевое слово** [*reserved word*] – лексема языка программирования, имеющая предопределенный смысл для транслятора. Ключевое слово никогда нельзя использовать в качестве идентификатора.

**Литерал** [*literal*] используется для записи *непосредственного значения*, например, чисел (числовые литералы – нумералы) и текстовых строк (символьные литералы).

## 4.2. Базовые синтаксические единицы исходного кода

**Фраза** [*phrase*] – любая последовательность символов, являющаяся правильной (валидной), то есть удовлетворяющая синтаксическим и семантическим правилам языка программирования.

**Выражение** [*expression*] – фраза на языке программирования, предназначенная для выполнения вычислений, которые позволяют получить некоторый результат (**значение выражения**). Выражение обычно состоит из *операндов* (переменных, констант и др.), объединенных *знаками операций* (вызов подпрограммы тоже можно считать операцией).

**Оператор** [*statement, operator*] – фраза на языке программирования, определяющая законченный этап обработки данных. В состав операторов входят ключевые слова, данные, выражения и др. Различают:

- 1) *атомарные* операторы, никакая часть которых не является самостоятельным оператором;
- 2) *структурные* операторы, объединяющие набор операторов в единый (укрупненный) оператор.

**Конструкция** [*construction, structure*] – структурный оператор, предназначенный для управления ходом выполнения программы.

**Объявление** [*declaration*] – фраза на языке программирования, связывающая идентификатор с объектом, то есть описание этого объекта, которое может включать информацию о типе, размере, значении, области видимости и других атрибутах. В большинстве блочно-структурированных языков программирования область видимости идентификатора определяется исключительно местом его объявления. Некоторые языки программирования (например, C++) разделяют понятия «объявление» и «определение» [*definition*].

## 4.3. Переменные и константы

**Переменная** [*variable*] – именованный элемент данных, значение которого может изменяться.

**Имя** [*name*] переменной – правильный идентификатор языка программирования, однозначно определяющий переменную.

Переменная содержит некоторое значение. Само по себе «значение» – базовое неопределяемое понятие.

**Значение** [*value*] переменной – значение элемента данных, который идентифицируется именем переменной.

**Тип** [*type*] переменной – тип данных, которые могут содержаться в переменной.

**Адрес** [*address*] переменной – адрес начала участка памяти, занимаемого значением переменной.

**Объявление** [*declaration*] переменной – первое упоминание переменной, задающее имя, тип и другие свойства переменной. Объявление может быть *явным*, то есть отдельным оператором, и *неявным*, являясь частью другого оператора.

Некоторые языки программирования разделяют понятия «объявление» и «определение» [*definition*] – сопоставление имени переменной конкретной области памяти и, соответственно, значения. Например, в языке C++ объявление подпрограммы совсем не то же самое, что её определение.

**Константа** [*constant*] – именованный элемент данных, значение которого задаётся при объявлении и затем **не может изменяться**. Имя константы не может появляться в левой части оператора присваивания.

#### 4.4. Реализация языка программирования (трансляторы)

**Машинный язык** [*machine language*] – язык, содержание и правила (синтаксис и грамматика) которого реализованы аппаратными средствами. Алфавит машинного языка большинства современных компьютеров состоит из двоичных цифр. Таким образом, текст программы на машинном языке состоит из последовательности бит и называется **машинным кодом**.

**Трансляция** программы [*translation*] – преобразование программы, представленной на одном из языков программирования, в программу на другом языке, в определенном смысле равносильная исходной.

**Транслятор** [*translator*] – программа, выполняющая трансляцию. Входными данными для транслятора является *исходная программа* на **исходном языке** [*source language*], а результатом – *целевая программа* на **целевом языке** [*target language*].

**Исходный код** [*source code*] – текст программы на исходном языке программирования.

**Единица трансляции** [*translation unit*] – часть текста программы, которая может транслироваться отдельно от остального исходного кода.

**Реализация языка** [*language implementation*] – создание транслятора этого языка, позволяющего исполнителю выполнять программу на этом языке. Существует два основных подхода к реализации языка (компиляция и интерпретация) и множество сочетаний.

**Компилятор** [*compiler*] – транслятор в машинный язык, на выходе которого непосредственно машинный код, пригодный к запуску на целевой архитектуре.

**Интерпретатор** [*interpreter*] – программа или среда программирования, в которой можно выполнить трансляцию исходного текста разрабатываемой программы и её выполнение без непосредственного получения машинного кода.

**Ассемблер** [*assembler*] – транслятор с языка ассемблера в машинные коды.

**Лексический анализатор** [*lexical analyzer, lexer*] – модуль, выполняющий лексический анализ исходного текста программы. Он преобразует исходный код в последовательность лексем, упрощая дальнейшие этапы компиляции.

**Синтаксический анализатор** или **модуль грамматического разбора** [*syntax analyzer, parser, parsing module*] – модуль, выполняющий проверку синтаксиса и преобразующий последовательность лексем в некоторое внутреннее представление, отражающее синтаксис входного языка.

**Генератор промежуточного кода** [*intermediate code generator*] – модуль, представляющий атрибутированное дерево грамматического разбора в линейном виде на некотором промежуточном языке. **Семантический анализатор** [*semantic analyzer*] – главный помощник генератора кода, определяющий смысл синтаксических конструкций и их семантическую правильность. Он выполняет следующие основные действия:

- 1) проверяет соблюдения семантических соглашений входного языка;
- 2) дополняет внутреннее представление программы в компиляторе атрибутами и действиями, неявно предусмотренными семантикой входного языка;
- 3) проверяет элементарные семантические (смысловые) нормы языка программирования, не связанные напрямую с входным языком.

**Оптимизатор** [*optimizer*] – модуль, выполняющий эквивалентные преобразования программы для повышения качества (обычно повышения скорости или уменьшения размера) машинного кода.

**Кодогенератор** [*code generator*] – модуль, преобразующий промежуточное представление (быть может, улучшенное оптимизатором) в машинный код. Очень часто это преобразование выполняется с использованием ассемблера, который становится модулем компилятора.

**Компоновщик** или **редактор связей** [*linker*] – модуль, который настраивает два или более компонента программы в машинном коде (связывает их) и обеспечивает возможность их одновременной загрузки и выполнения.

**Директива транслятора/компилятора** [*translator/compiler directive*] – фраза на языке программирования, не являющаяся оператором или объявлением, но распознаваемая транслятором как инструкция изменения параметров его работы.

## 4.5. Модуляризация

**Модуляризация** [*modularization*] – членение системы на относительно независимые компоненты с чётко определёнными интерфейсами.

Декомпозиция на модули – универсальный механизм, используемый в технике повсеместно. Единицей декомпозиции исходного кода программ является



**программный модуль** [*program module, unit*] – специально оформленная часть исходного кода, которая может являться единицей хранения, трансляции, объединения с другими модулями и загрузки в оперативную память.

**Интерфейс модуля** [*module interface*] – часть модуля, содержащая объявления типов, констант, переменных и сигнатур подпрограмм, которые определены в модуле и могут использоваться его клиентами.

Синтаксически модуляризация может выполняться различными способами. Сравните, например, языки *Fortran, C, D, Pascal, Ada, Python, C#*.

## Раздел 5. Подпрограммы

**Подпрограмма** [*subroutine*] – именованная часть программы, которая разрабатывается относительно независимо от других частей и может быть многократно вызвана (выполнена) по своему имени. Исходный код, включённый в подпрограмму, называется её **телом** [*body*].

**Формальный параметр** [*formal parameter*] – параметр, понимаемый как его описание в заголовке подпрограммы при её объявлении. Прилагательное «формальный» обычно опускается.

**Фактический параметр** [*actual parameter*] или **аргумент** [*argument*] – выражение, подставленное вместо формального параметра при вызове подпрограммы.

**Передача параметров** [*parameters passing*] – интерпретация того, что происходит при подстановке фактических параметров на место формальных в момент вызова подпрограммы.

**Параметр по умолчанию** [*default parameter*] – параметр подпрограммы, для которого можно не указывать аргумент при вызове подпрограммы. В этом случае он принимает значение *по умолчанию*, которое указывается при объявлении подпрограммы.

**Процедура** [*procedure*] – подпрограмма, вызов которой является отдельным оператором.

**Функция** [*function*] – подпрограмма, вычисляющая и возвращающая значение, благодаря чему её вызов является выражением.

**Соглашение о вызове** [*calling convention*] определяет и формализует в терминах исполнимого кода целевого компьютера использование памяти, стека и регистров процессора, ответственность за выделение и освобождение памяти под аргументы, порядок аргументов в памяти, стеке или регистрах ЦП.

**Сигнатура** [*signature*] подпрограммы – информация, достаточная для правильной организации вызова подпрограммы. В сигнатуру входит соглашение о вызове и описание всех формальных параметров (включая возвращаемое значение в случае функции).

**Локальная (вложенная) подпрограмма** [*nested subroutine*] – подпрограмма, объявление которой находится внутри объявления другой подпрограммы, которая становится её областью видимости.

**Побочный эффект** [*side effect*] – изменение при выполнении подпрограммы глобальных переменных, не являющихся аргументами подпрограммы.

**Перегрузка** [*subroutine overloading*] – наличие в рамках одного пространства имён разных подпрограмм с *одинаковыми именами*, если их списки параметров различаются.

**Предусловие** [*precondition*] – логическое условие, которое должно соблюдаться *перед* вызовом подпрограммы.

**Постусловие** [*postcondition*] – логическое условие, которое должно соблюдаться *после* завершения работы подпрограммы.

**Инвариант** [*invariant*] – условие, которое должно соблюдаться *постоянно* (по меньшей мере, до и после некоторого действия). Инварианты особенно популярны при выполнении циклических действий. Инвариантом цикла называют условие, выполняющееся после каждой итерации цикла.

## Раздел 6. Данные и их структуры

**Данные** в общем виде (по ГОСТ Р ИСО 10303-1-99) – информация, представленная в формальном виде, пригодном для передачи, интерпретации или обработки людьми или компьютерами. Сокращая объём термина, данными назовём информацию, представленную в *формализованном виде*, пригодном для автоматизированной или автоматической обработки *компьютерами*. Информационные процессы, использующие данные, называются **обработкой данных** [*data processing*].

Обработка данных разбивается на несколько областей.

1. Ввод/вывод – извлечение информации из внешней среды и её формализация в виде данных или сохранение данных во внешней среде.
2. Преобразование форматов – изменение способа представления данных.
3. Передача (телекоммуникации) – передача данных между компонентами распределённых информационных систем.
4. Хранение – обеспечение долговременной доступности, целостности и защищённости данных.
5. Поиск – выполнение поисковых запросов к массивам данных.

б. Переработка – существенное преобразование содержания и/или формы данных, выполняемое на основе анализа и/или синтеза.

**Система данных** [*data system*] – взаимосвязанная структурированная совокупность данных, обладающая свойствами, отсутствующими у любых подмножеств этих данных по отдельности. Несистематизированные данные никого не интересуют.

**Структура данных** [*data structure*] – структура связей между элементами в системе представления данных. Таблица 6.1 иллюстрирует основные уровни рассмотрения систем аналитиком предметной области, алгоритмистом/прикладным математиком, прикладным программистом и системным программистом.

Таблица 6.1. Уровни рассмотрения систем и структур данных

| Уровень системы/структуры данных   | Объект рассмотрения  | Связываемые понятия                               |
|------------------------------------|--|---|
| Концептуальный                     | Решаемая задача  | Понятия предметной области                        |
| Логический 1<br>(алгоритмизация)   | Алгоритм   | Абстрактные типы данных<br>(формализация понятий) |
| Логический 2<br>(программирование) | Код программы<br>(исходный код на языке<br>программирования) | Конкретные типы данных<br>(реализация понятий)    |
| Физический                         | Программа<br>(во время исполнения)                           | Участки памяти<br>(работа исполнителя)            |

**Информационный элемент** [*data entry*] – любой логический элемент структуры данных.

**Атомарный** [*atomic, simple*] информационный элемент (ячейка) – информационный элемент, не имеющий собственной логической структуры и представляющий атомарное значение.

**Формат данных** [*data format*] – совокупность правил представления и интерпретации данных в памяти компьютера, на внешних носителях, при операциях ввода/вывода и при передаче по каналам связи.

**Тип данных** (конкретный тип) [*data type*] – характеристика, явно или неявно приписываемая объекту (переменной, константе, полю записи, выражению, функции и т. п.) и определяющая множество допустимых значений, формат хранения данных, размер выделяемой под них памяти и набор операций, которые над ними можно производить.

**Система типов** [*type system*] языка программирования определяет состав базовых типов языка, способы создания новых типов и правила их использования.

**Контроль соответствия типов** [*type checking*] – проверка того, что тип переменной совместим с типом выражения при выполнении оператора присваивания, при подстановке аргумента на место формального параметра и в других случаях сопоставления различных данных между собой.

**Приведение типов** [*type casting*] – явное (указанное программистом) преобразование типа отдельной переменной или выражения в другой тип на время выполнения отдельной операции.

**Атомарный (простой) тип** [*atomic/simple type*] описывает тип атомарного информационного элемента на физическом уровне.

**Составной (комплексный) тип** [*complex/compound type*] – тип данных, образованный путём объединения в некоторую структуру набора простых типов.

**Кортеж** [*tuple*] длины  $n$  (также **упорядоченный набор длины  $n$**  или  **$n$ -ка**) – упорядоченная последовательность ровно из  $n$  элементов любой природы (где  $n$  – натуральное число). Отметим, что  $n$ -ками чаще всего обозначают короткие кортежи: «пара», «тройка» и т.д. В математике индексы элементов кортежа называют *координатами*. Кортежи часто становятся основой построения составных типов.

**Приоритет** [*precedence*] – характеристика операции, определяющая порядок выполнения операций в сложных выражениях.

**Тип данных с управляемым временем жизни** [*lifetime-managed type*] – тип данных, переменные которого являются динамическими, но используют память без явного контроля со стороны программиста.

## 6.1. Индексы и ссылки

**Ссылка** [*reference*] (в общем) – информационный элемент, связывающий между собой некоторые данные. Другими словами, ссылка – такой информационный элемент, который указывает на источник данных вместо хранения их самих. Такой информационный элемент часто называют **связью** [*link*].

**Индекс** [*index*] – порядковый номер элемента. Индекс сам по себе не является информационным элементом!

**Массив** [*array*] – последовательность однотипных элементов, к каждому из которых можно получить доступ по его индексу за константное время (примерно одинаковое время, не зависящее от значения индекса). Массив – базовая структура данных с индексацией.

**Указатель** [*pointer*] – адрес элемента, используемый в качестве ссылки. Такая связь абсолютна.

**Курсор** [*cursor*] – индекс, используемый в качестве ссылки. Такая связь относительна и требует знания адреса начального элемента массива.

**Список** [*list*] – последовательность однотипных элементов, каждый из которых связан с соседними (одним или двумя) посредством ссылок. Список – базовая ссылочная структура.

## 6.2. Абстрактные типы данных

### 6.2.1. Формализация абстрактного типа данных

**Абстрактный тип данных** (АТД) [*abstract logic design, abstract data type (ADT)*] – функциональное описание некоторого класса сущностей в терминах операций, которые могут выполняться над ними.

**Интерфейс АТД** [*ADT interface*] – формальное и однозначное описание синтаксиса и семантики операций, которые могут выполняться над экземплярами АТД. Так же, как описание языка программирования не определяет особенности его *реализации*, так и интерфейс АТД не определяет *реализацию АТД*, но может задавать некоторые её характеристики, например, верхнюю оценку вычислительной сложности (см. раздел 9).

**Реализация АТД** [*ADT implementation*] – конкретный тип данных, обеспечивающий заданный интерфейс АТД.

### 6.2.2. Контейнеры

**Контейнер** [*container*] – абстрактный тип данных, представляющий собой структурированную коллекцию информационных элементов, доступ к которым определяется структурой контейнера. По умолчанию считают, что информационные элементы контейнера имеют *один и тот же тип*. Если это не так, то такое свойство контейнера специально оговаривается.

Добавление и удаление элементов контейнера назовём его **трансформацией**. Доступ к элементу контейнера – операция получения или изменения значения этого элемента.

**Последовательность** [*sequence*] – контейнер, в котором элементы упорядочены по индексам (пронумерованы). Не ограничивая общности, можно считать, что индексами элементов последовательностей являются целые числа от 0 до  $n-1$ , где  $n$  – число элементов последовательности (размер). Элементы с индексами 0 и  $n-1$  будем называть **концевыми**. Элемент с индексом 0 будем называть **начальным** элементом, элемент с индексом  $n-1$  – **конечным**. Если последовательность пуста, то концевые элементы не определены. Если размер последовательности равен 1, то её единственный элемент является одновременно и начальным, и конечным.

Для последовательности определены операции добавления элемента до/после элемента с заданным индексом и удаления элемента с заданным индексом. Если последовательность пуста, то любая операция добавления элемента приводит к одному и тому же результату – появлению элемента с индексом 0. Обычно операция удаления возвращает значение удалённого элемента.

**Вектор** [*vector*] – последовательность, в которой возможен доступ к любому элементу по *индексу* элемента.

**Универсальный вектор** [*universal vector*] – вектор, в котором возможно добавление/удаление произвольных элементов.

**Статический вектор** [*static vector*] – вектор, в котором возможна только установка *размера вектора*, которую можно трактовать как удаление всех элементов лексикос последующим добавлением заданного количества, но невозможно удаление/добавление произвольных элементов.

**Кортеж** [*tuple*] длины  $n$  – упорядоченная последовательность ровно из  $n$  элементов *произвольного типа*. В терминологии многих языков программирования кортежом часто называют *неизменяемый статический вектор*.

**Дек** [*deque*] – последовательность, в которой возможны только:

- 1) доступ: к конечным элементам;
- 2) добавление: до начального и после конечного элемента;
- 3) удаление: конечных элементов.

**Стек** [*stack*] – дек, в котором возможны только:

- 1) доступ: к конечному элементу (**вершине стека** [*top*]);
- 2) добавление: после конечного элемента;
- 3) удаление: конечного элемента.

**Очередь** [*queue*] – дек, в котором возможны только:

- 1) доступ: к начальному элементу (**голове очереди** [*head*]);
- 2) добавление: после конечного элемента (**хвоста очереди** [*tail*]);
- 3) удаление: начального элемента.

**Очередь с приоритетами** (приоритетная очередь) [*priority queue*] – последовательность, в которой каждому элементу назначается приоритет и возможны только:

- 1) доступ: к элементу с наибольшим приоритетом;
- 2) добавление: после конечного элемента;
- 3) удаление: элемента с наибольшим приоритетом.

**Множество** [*set*] – АД, соответствующий математическому определению множества, но, как правило, однотипных элементов. Для множества стандартными операциями являются:

- 1) проверка того, что элемент с данным значением принадлежит множеству;
- 2) добавление элемента (если элемент с данным значением уже принадлежит множеству, то ничего не происходит);
- 3) удаление элемента (если элемент с данным значением не принадлежит множеству, то ничего не происходит);
- 4) нахождение объединения, пересечения и разности с другим множеством.

**Коллекция** [*collection*] – множество элементов произвольных типов.

**Мультимножество** (множество с повторениями, сумка) [*multiset, bag*] – расширение понятия множества, позволяющее включать несколько одинаковых элементов. Другими словами, каждому типу элемента соответствует атрибут «количество элементов этого типа».

**Словарь** [*dictionary*] или **отображение** [*map*] – ассоциативный контейнер, элементы которого разделены на **ключевое поле** (ключ) и **информационное поле**, и доступ к элементам возможен по значению ключевого поля. Для словаря стандартными операциями являются:

- 1) проверка того, что в словаре есть элемент с данным значением ключа;
- 2) доступ к информационному полю элемента с заданным значением ключа;
- 3) добавление элемента (если элемент с данным значением ключа уже присутствует, то его информационное поле изменяется);
- 4) удаление элемента (если элемент с данным значением ключа отсутствует, то ничего не происходит);

Можно считать, что словарь является расширением понятия множества. Словарь по сути становится мультимножеством ключей, если информационное поле – целого типа.

**Упорядоченный словарь** [*ordered dictionary*] – словарь, на множестве *ключевых полей* которого задан линейный порядок и возможно перечисление элементов в соответствии с этим порядком.

**Дважды упорядоченный словарь** [*double-ordered dictionary*] – упорядоченный словарь, на множестве *информационных полей* которого также задан линейный порядок и возможно перечисление элементов в соответствии с ним.

### 6.3. Структуры данных как механизм реализации АД

**Массив** [*array*] и **связный список** [*linked list*] (как структуры данных) – соответственно базовые индексная и ссылочная структуры данных в памяти (см. раздел 6.1). *Массив* представляет собой непрерывный участок памяти, а *список* – кусочки памяти для каждого элемента, связанные указателями (ссылками в виде

адресов). Массив идентифицируется адресом первого элемента. Список идентифицируется адресом одного из своих элементов, от которого по указателям можно добраться до всех остальных элементов.

Все более сложные структуры данные представляют собой «переплетение» набора массивов и списков. Действительно, как элементом массива, так и информационной частью элемента списка может быть адрес другого массива или элемента списка.

**Хэш-таблица** [*hash table*] – структура данных для реализации неупорядоченных ассоциативных контейнеров с константным (при соблюдении некоторых условий) временем доступа, добавления и удаления произвольных элементов. Эта структура может состоять не только из *таблицы*, но таблица в виде массива – основная её часть, необходимая для реализации *отображения*. Таблица может сама содержать информационные элементы или являться оглавлением для них, а её размер должен быть значительно (примерно в два раза) больше текущего числа хранимых элементов для обеспечения константного времени выполнения основных операций.

**Хэш-функция** [*hash function*] (здесь) – эффективно вычисляемая функция, возвращающая для аргумента, тип которого соответствует типу элементов хэш-таблицы, номер из диапазона индексов ячеек хэш-таблицы. «Хорошая» хэш-функция должна обладать, кроме низкой вычислительной сложности (высокой скорости вычисления), равномерностью покрытия индексов и хорошей *перемешиваемостью*.

**Дерево** [*tree*] – нелинейная ссылочная структура данных, каждый элемент которой связан с другими цепочками ссылок, образующими древовидную структуру (корневое дерево в теоретико-графовом смысле).

**Узел** [*node*] **дерева** – любой его элемент. Узел состоит из *информационной части* и набора *ссылок* на другие узлы.

**Путь в дереве** [*path*] – последовательность неповторяющихся узлов ( $n_1, \dots, n_k$ ), такая что существует ссылка из  $n_i$  в  $n_{i+1}$ , где  $i = 1 \dots k-1$ . **Длина пути** в дереве – число *ссылок*, по которым нужно пройти, чтобы последовательно посетить все узлы в пути.

**Расстояние** между узлами  $n_1$  и  $n_2$  в дереве – минимальная длина пути из  $n_1$  в  $n_2$ . Расстояние между узлами, которые не соединены путём, *не определено*.

**Корень дерева** (корневой узел) [*root*] – логически выделенный узел, от которого по ссылкам можно перейти к любому другому узлу, и на который обязательно существует ссылка извне.

**Лист дерева** (листовой узел) [*leaf*] – узел, не ссылающийся на элементы, находящиеся дальше от корня, чем он сам.

**Предок** узла [*ancestor*] узла  $n$  – любой узел кроме  $n$ , входящий в путь от корня до  $n$ . Корень не имеет предков.



**Потомок** узла [*descendant*] узла  $n$  – любой узел кроме  $n$ , входящий в путь, не проходящий через корень, от  $n$  до любого листа. Лист не имеет потомков.

**Родительский узел** [*parent*] узла  $n$  – узел, удовлетворяющий двум условиям: 1) он входит в путь от корня до  $n$ ; 2) он содержит ссылку на  $n$ .

**Дочерний узел** [*child*] узла  $n$  – узел, удовлетворяющий двум условиям: 1) он входит в путь от  $n$  до листа; 2)  $n$  содержит ссылку на него.

**Степень** [*degree*] узла – число дочерних узлов. Степень листа равна нулю.

**Поддерево** узла  $n$  [*subtree*] – дерево, состоящее из какого-либо дочернего узла  $n$  (корня поддерева) и всех его потомков. Число различных поддеревьев узла равно степени узла.

**Высота дерева** [*height*] – максимальная из длин путей от корня до листа.

**Сбалансированное дерево** [*balanced tree*] – дерево, у которого длины путей от корня до всех листьев отличаются *незначительно*.

**AVL-сбалансированное дерево** [*AVL-balanced tree*] – бинарное дерево, у которого высоты поддеревьев любой вершины отличаются не более, чем на единицу.

**Самобалансирующееся дерево** [*self-balanced tree*] – дерево, которое остаётся сбалансированным после добавления/удаления узлов, для чего применяется априори заданная для конкретного дерева *процедура балансировки*.

## Раздел 7. Объектно-ориентированное проектирование и конструирование

### 7.1. Концепции объектно-ориентированного подхода

**Объектно-ориентированный подход** [*object-oriented approach*] (ООП) – это подход к решению проблем, предполагающий *объектовую* (базовое понятие) декомпозицию проблемы, в отличие, например, от процедурной декомпозиции.

**Объектно-ориентированное программирование** [*object-oriented programming*] – это методология (парадигма) программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром некоторого класса, входящего в иерархию наследования.

#### 7.1.1. Объекты и отношения между ними

**Объект** [*object*] – произвольная сущность, для которой достаточно легко определить *границы, состояние и поведение*.

**Атрибут** [*attribute*] – свойство объекта. Совокупность значений атрибутов определяет состояние объекта.

**Метод** [*method*] – вариант поведения объекта. Совокупность методов определяет всё поведение объекта.

**Интерфейс** [*interface*] объекта – внешнее представление объекта, то есть все его атрибуты и методы. Описание интерфейса задаёт границы объекта самим фактом включения или не включения тех или иных атрибутов или методов.

**Класс** [*class*] – описание однотипных объектов, то есть их интерфейса. Класс можно представить в виде шаблона, описывающего возможные состояния и поведение объектов данного класса. Объект, удовлетворяющий этому шаблону, называется **экземпляром** [*instance*] класса. Объявление класса формализует границы, состояние и поведение, о которых идёт речь в определении объекта. Все экземпляры класса имеют *один и тот же* интерфейс. Объект связан с классом отношением «это есть» [*is-a*].

**Инкапсуляция** [*encapsulation*] – объединение более простых сущностей в более сложную с одновременным их сокрытием за интерфейсом новой сущности, а также механизм такого сокрытия. Объектная инкапсуляция, в отличие от инкапсуляции структур данных и процедурной инкапсуляции, скрывает за интерфейсом класса (атрибутами и методами) *одновременно* данные и алгоритмы.

**Наследование** [*inheritance*] – способность класса получить в своё собственное распоряжение состояние и поведение другого класса, не прибегая к агрегации, а также механизм обеспечения этой способности.

**Полиморфизм** [*polymorphism*] – способность наследников класса различным образом реализовывать определяемое его интерфейсом поведение, а также механизм обеспечения этой способности.

**Агрегация** [*aggregation*] является отражением отношения «являться частью» или «часть/целое» [*has-a*]. Агрегация подразумевает включение одного класса в другой в виде *атрибута*.

**Композиция** [*composition*] – вариант агрегации, когда «целое» управляет временем жизни «части».

## Раздел 8. Информационный поиск и сортировка

**Ключ** [*key*] – набор атрибутов объекта, используемый при формализации некоторого отношения между объектами. Ключ называется *простым*, если он представляет собой один атомарный атрибут, в противном случае ключ называется *составным*.

**Лексикографический порядок** [*lexicographical order*] – порядок на кортежах, задаваемый следующим бинарным отношением предшествования. Пусть *a* и *b* – два произвольных кортежа (слова), составленных из элементов множества *A* (алфавита),

на котором определено отношение порядка. Слово  $a$  *предшествует* слову  $b$  (обозначается  $a < b$ ), тогда и только тогда, когда имеет место один из следующих случаев.

1.  $a = vxc$ ,  $b = vud$ , где  $v$ ,  $c$  и  $d$  – некоторые (возможно пустые) слова,  $x, y \in A, x < y$ .
2.  $b = av$ , где  $v$  – некоторое непустое слово.

**Информационный поиск** [*data search, searching*] – процесс нахождения в хранилище информации некоторых информационных элементов, удовлетворяющих заданным условиям, или определения их свойств, необходимых пользователю.

**Трасса поиска** [*search trace*] – элементы контейнера, анализируемые в процессе поиска. При последовательной реализации алгоритма поиска это последовательность проверяемых элементов.

**Сортировка** [*sorting*] – упорядочение набора некоторых объектов по некоторому критерию. Обычно критерий является формализацией отношения на *ключах*.

**Оглавление** (индексный массив) [*index array*] – массив курсоров, то есть индексов элементов некоторой последовательности.

## Раздел 9. Вычислительная сложность

**Временная сложность** [*time complexity*] – оценка времени работы алгоритма в виде функции от некоторых характеристик входных данных.

**Емкостная (пространственная) сложность** [*space complexity*] – оценка объёма памяти, необходимого для работы алгоритма в виде функции от некоторых характеристик входных данных.

**Размер входа** [*input size*] – длина последовательности бит, которая описывает входные данные задачи.

**Размер выхода** [*output size*] – длина последовательности бит, которая описывает результат решения задачи.

**Задача распознавания свойств** [*decision problem*] – задача, размер выхода которой равен **1** биту, то есть которая может быть сформулирована в виде вопроса, на который нужно ответить «да» или «нет».

**Асимптотическая оценка** [*asymptotical estimate*] временной и емкостной сложности алгоритмов решения задач распознавания свойств в зависимости от размера входа  $x$  (поскольку время выполнения и объём памяти – заведомо неотрицательные величины, то можно не указывать, что рассматривается модуль значения функций):

- 1)  $t(x) = O(f(x))$  (читается как «О-большое» [Big-O])  $\Leftrightarrow$  существуют константы  $C$  и  $a$ , такие что  $t(x) \leq C(f(x))$  для любого  $x > a$ ;
- 2)  $t(x) = \Omega(f(x))$   $\Leftrightarrow$  существуют константы  $C$  и  $b$ , такие что  $t(x) \geq C(f(x))$  для любого  $x > b$ ;
- 3)  $t(x) = \Theta(f(x))$   $\Leftrightarrow$  одновременно  $t(x) = O(f(x))$  и  $t(x) = \Omega(f(x))$ .

## Раздел 10. Рекомендуемые источники информации

### 10.1. Основные публикации

#### 10.1.1. Общие вопросы

1. Жилин Д.М. Теория систем: опыт построения курса. – М.: КомКнига, 2006. – 184 с.
2. Дейкстра Д. Дисциплина программирования. – М.: Мир, 1985. – 274 с.

#### 10.1.2. Инженерия ПО

3. Макконнелл С. Профессиональная разработка программного обеспечения. – М.: Символ-Плюс, 2006. – 240 с.
4. Брукс Ф. Мифический человеко-месяц или Как создаются программные системы. – Символ-Плюс, 2006 г. – 304 с.
5. Спольски Д. Джоэл о программировании. – Символ-Плюс, 2006. – 352 с.
6. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – СПб.: Бином, Невский Диалект, 1998. – 560 с.
7. Буч Г., Рамбо Дж., Якобсон И. Язык UML. Руководство пользователя. – М.: ДМК пресс, 2007. – 496 с.
8. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с.

#### 10.1.3. Математика для программистов (дискретная)

9. Новиков Ф.А. Книга Дискретная математика для программистов. 3-е изд. Питер, 2008. 384 с.
10. Грэхем Р., Кнут Д., Паташник О. Конкретная математика. Основание информатики. Мир, 1998. 703 с.
11. Зыков А.А. Основы теории графов. Вузовская книга, 2004. 664 с.
12. Айгнер М. Комбинаторная теория. Мир, 1982. 558 с.
13. Гэри М., Джонсон Д., и др. Вычислительные машины и труднорешаемые задачи. Мир, 1982. 420 с.
14. Хоар Ч.Э. Взаимодействующие последовательные процессы. Мир, 1989. 264 с.

#### 10.1.4. Построение эффективных алгоритмов

15. Левитин А.В. Алгоритмы: введение в разработку и анализ. Вильямс, 2006. 576 с.
16. Вирт Н. Алгоритмы и структуры данных. Невский Диалект, 2001. 352 с.
17. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос. Вильямс, 2000. 384 с.
18. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 3-е изд. Вильямс, 2013. 1328 с.
19. Седжвик Р. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных. Вильямс, 2013. 1056 с.
20. Седжвик Р. Фундаментальные алгоритмы на C++. Части 1-4: Анализ, структуры данных, сортировка, поиск (3-я редакция). ООО «ДиаСофтЮП», 2002. 688 с.

21. **Седжвик Р.** Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ. ООО «ДиаСофтЮП», 2002. 496 с.
22. **Седжвик Р.** Фундаментальные алгоритмы на C. Части 1-4: Анализ/ Структуры данных/Сортировка/Поиск. ООО «ДиаСофтЮП», 2003. 672 с.
23. **Седжвик Р.** Фундаментальные алгоритмы на C. Часть 5: Алгоритмы на графах. ООО «ДиаСофтЮП», 2003. 480 с.
24. **Макконнелл Дж.** Основы современных алгоритмов. 2006. 368 с.
25. **Кнут Д.** Искусство программирования, том 1. Основные алгоритмы, 3-е изд. Вильямс, 2000. 720 с.
26. **Кнут Д.** Искусство программирования, том 2. Получисленные алгоритмы, 3-е изд. Вильямс, 2000. 832 с.
27. **Кнут Д.** Искусство программирования, том 3. Сортировка и поиск, 2-е изд. Вильямс, 2000. 832 с.
28. **Липский В.** Комбинаторика для программистов. Мир, 1988. 213 с.
29. **Зубов В.С., Шевченко И.В.** Структуры и методы обработки данных. Практикум в среде Delphi. ФИЛИНЪ, 2004. 304 с.
30. **Бакнелл Джулиан М.** Фундаментальные алгоритмы и структуры данных в Delphi. ООО «ДиаСофтЮП», 2003. 560 с.
31. **Касьянов В.Н., Евстигнеев В.А.** Графы в программировании: обработка, визуализация и применение. БХВ-Петербург, 2003. 1104 с.
32. Алгоритмы и программы решения задач на графах и сетях / **Нечепуренко М.И., Попков В.К., Кохов В.А. и др.** Новосибирск: Наука, 1990. 515 с.

#### ***10.1.5. Программирование (конструирование программных продуктов)***

33. **Себеста Р.** Основные концепции языков программирования. Вильямс, 2001. 672 с.
34. **Макконнелл С.** Совершенный код. Практическое руководство по разработке программного обеспечения. Питер, 2005. 896 с.
35. **Хант Э., Томас Д.** Программист-прагматик. Лори, 2004. 270 с.
36. **Фаулер М.** Рефакторинг. Улучшение существующего кода. Символ-Плюс, 2005. 432 с.
37. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. 3 издание. ДМК, 2011. 376 с.

#### ***10.1.6. Доступ к данным в программных системах***

38. **Кренке Д.** Теория и практика построения баз данных. Питер, 2005. 864 с.
39. **Дейт К.** Введение в системы баз данных, 8-е издание. Вильямс, 2005. 1328 с.
40. **Дейт К., Дарвен Х.** Основы будущих систем баз данных. Третий манифест. Янус-К, 2004. 656 с.

#### ***10.1.7. Программирование в ОС Microsoft Windows***

41. **Рихтер Дж.** Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows, 4-е изд. Питер, 2001. 752 с.
42. **Рихтер Дж.** Программирование на платформе Microsoft .NET Framework. Мастер-класс. Питер, 2005. 512 с.

### 10.1.8. Системное программирование

43. Ахо А., Сети Р., Ульман Дж.Д., Лам М. Компиляторы: принципы, технологии и инструменты. Вильямс, 2008. 1184 с.
44. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. 3 издание. СПб.: Питер, 2006. 958 с.
45. Таненбаум Э., Ван Стен М. Распределённые системы. Принципы и парадигмы. Питер, 2003. 877 с.
46. Intel® 64 and IA-32 Architectures Software Developer Manuals. Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C. Intel, January 2015. 3463 p.

### 10.2. Некоторые Интернет-источники

47. *Dictionary of Algorithms and Data Structures by NIST* (<http://xlinux.nist.gov/dads/>)
48. *Intel® 64 and IA-32 Architectures Software Developer Manuals* (<http://software.intel.com/en-us/articles/intel-sdm>)
49. *AMD Developer Guides, Manuals & ISA Documents* (<http://developer.amd.com/resources/developer-guides-manuals/>)
50. *List of programming languages by type* ([http://en.wikipedia.org/wiki/List\\_of\\_programming\\_languages\\_by\\_type](http://en.wikipedia.org/wiki/List_of_programming_languages_by_type))
51. *Computer Languages History by Éric Lévénez* (<http://www.levenez.com/lang/>)
52. [Old] *HOPPL: an interactive Roster of Programming Languages* (<http://web.archive.org/web/20111205165034/http://hopl.murdoch.edu.au/>)
53. *The Stony Brook Algorithm Repository by Steven Skiena* (<http://algorist.com/algorist.html>)
54. *Computer Programming Algorithms Directory* (<http://www.algosort.com>)
55. *(Data Structures and Algorithms)*
56. *Algorithms and Data Structures Research & Reference Material by Godfried Toussaint* (<http://cgm.cs.mcgill.ca/~godfried/teaching/algorithms-web.html>)
57. *Guide to the Systems Engineering Body of Knowledge (SEBoK)* (<http://www.sebokwiki.org>)
58. *Software Engineering Body of Knowledge (SWEBoK)* (<http://www.computer.org/education/bodies-of-knowledge/software-engineering>)
59. Сравнение языков программирования ([http://ru.wikipedia.org/wiki/Сравнение\\_языков\\_программирования](http://ru.wikipedia.org/wiki/Сравнение_языков_программирования))
60. Интернет-университет информационных технологий (<http://www.intuit.ru>)
61. Дискретная математика, алгоритмы и структуры данных, сайт ИТМО ([http://neerc.ifmo.ru/wiki/index.php?title=Дискретная\\_математика,\\_алгоритмы\\_и\\_структуры\\_данных](http://neerc.ifmo.ru/wiki/index.php?title=Дискретная_математика,_алгоритмы_и_структуры_данных))
62. Информатика и программирование шаг за шагом, сайт кафедры Информационных технологий Курганского гос. Университета (<http://it.kgsu.ru>)
63. Алгоритмы и методы, сайт Ильи Кантора (<http://algotlist.manual.ru>)
64. Сайт Джоэла Сполски (<http://www.joelonsoftware.com>)
65. Сервис онлайн тестирования *Quizful* (<http://www.quizful.net>)

## Раздел 11. Список общепринятых аббревиатур

1. -- [API] – Интерфейс Прикладного Программирования
2. АДД [ADT] – Абстрактный Тип Данных
3. ОО [OO] – Объектно-Ориентированный (как часть аббревиатуры)
4. ОП [MM] – Оперативная Память
5. ОС [OS] – Операционная Система
6. ПК [PC] – Персональный Компьютер
7. ПО [SW] – Программное Обеспечение
8. ПП [-] – Программный Продукт
9. ЦП [CPU] – Центральный Процессор
10. ЯПВУ [HLPL] – Язык Программирования Высокого Уровня



## Раздел 12. Терминологический указатель

- Абстрактная машина, 9
- Абстрактный тип данных. *See* Тип данных:Абстрактный
- Абстракция, 3
- Уровень абстракции, 3
- Агрегация, 26
- Адрес переменной, 14
- Алгоритм, 4
- Алфавит, 13
- Аргумент. *See* Параметр:Фактический
- Архитектура компьютера, 8
- Архитектура программной системы, 5
- Асимптотическая оценка сложности, 27
- Ассемблер, 15
- Ассемблера язык. *See* Язык:Ассемблера
- Атрибут, 25
- Байт, 9
- Бит, 9
- Вектор, 22
- Статический, 22
- Универсальный, 22
- Вершина стека, 22
- Выравнивание, 10
- Выражение, 14
- Высота дерева, 25
- Генератор промежуточного кода, 16
- Голова очереди, 22
- Грамматика, 12
- Данные, 4, 18
- Дек, 22
- Дерево, 24
- AVL-сбалансированное, 25
- Самобалансирующееся, 25
- Сбалансированное, 25
- Директива транслятора, 16
- Дистрибутив, 7
- Длина пути в дереве, 24
- Документ, 7
- Дочерний узел, 25
- Единица трансляции, 15
- Жизненный цикл, 7
- ЖЦПО. *See* Жизненный цикл
- Задача распознавания свойств, 27
- Значение переменной, 14
- Идентификатор, 13
- Имя переменной, 14
- Инвариант, 18
- Индекс, 20
- Инкапсуляция, 26
- Интерпретатор, 15
- Интерфейс, 3
- Абстрактного типа данных, 21
- Модуля, 17
- Объекта, 26
- Информационный поиск, 27
- Информационный элемент, 19
- Атомарный, 19
- Инфраструктура, 5
- Класс, 26
- Ключ, 26, *См.* Поле:Ключевое
- Ключевое слово, 13
- Код, 4
- Исходный, 15
- Кодогенератор, 16
- Коллекция, 23
- Комментарий, 13
- Компилятор, 15
- Композиция, 26
- Компоновщик, 16
- Конвейеризация, 9
- Конечный пользователь, 7
- Константа, 15
- Конструирование, 4
- Конструкция, 14
- Контейнер, 21
- Контроль соответствия типов, 20
- Корень дерева, 24
- Кортеж, 20

- как АДТ, 22
- Курсор, 21
- Кэширование, 9
- Кэш-память, 9
- Лексема, 13
- Лексикографический порядок, 26
- Лексический анализатор, 16
- Лист дерева, 24
- Литерал, 13
- Лицензионное соглашение, 7
- Массив, 20
- Машинное слово. *См.* Слово:Машинное
- Машинный код, 15
- Машинный язык. *See* Язык:Машинный
- Метка, 13
- Метод, 26
- Множество, 23
- Модель вычислений, 10
- Модуль грамматического разбора. *See* Синтаксический анализатор
- Модуляризация, 16
- Морфология, 12
- Мультимножество, 23
- Наследование, 26
- Область видимости, 13
- Область действия. *See* Область видимости
- Обработка данных, 18
- Объект, 25
- Объектно-ориентированный подход, 25
- Объявление, 14
- Объявление переменной, 15
- Оглавление, 27
- Оператор, 14
- Оптимизатор, 16
- Организация компьютера. *See* Структура:Компьютера
- Отображение. *See* Словарь
- Очередь, 22
  - С приоритетами, 22
- Память, 8
  - Оперативная, 8
- Парадигма, 10
- Парадигма программирования
  - Декларативная, 10
  - Императивная, 10
  - Объектно-ориентированная, 11
- Параметр
  - По умолчанию, 17
  - Фактический, 17
  - Формальный, 17
- Перегрузка подпрограмм, 18
- Передача параметров, 17
- Переменная, 14
- Платформа, 6
  - Развёртывания, 6
  - Разработки, 6
- ПО. *See* Программное обеспечение
- Побочный эффект, 18
- Поддерево, 25
- Подпрограмма, 17
  - Вложенная. *See* Подпрограмма:Локальная
  - Локальная, 18
- Поиск. *See* Информационный поиск
- Поле
  - Информационное, 23
  - Ключевое, 23
- Полиморфизм, 26
- Последовательность, 21
- Постусловие, 18
- Потомок узла дерева, 25
- ПП. *See* Программный продукт
- Прагматика, 12
- Предок узла дерева, 24
- Предусловие, 18
- Приведение типов, 20
- Приоритет операции, 20
- Приоритетная очередь. *See* Очередь:С приоритетами
- Программа, 4
- Программирование, 4
  - Декларативное, 11
  - Императивное, 11
  - Логическое, 12

- Объектно-ориентированное, 25
- Структурное, 11
- Функциональное, 12
- Программное обеспечение, 4
- Программный продукт, 4
- Производительность, 6
- Процедура, 17
- Процессор, 8
  - Центральный, 8
- Путь в дереве, 24
- Размер входа, 27
- Размер выхода, 27
- Разрядность процессора, 9
- Распараллеливание, 9
- Реализация
  - Абстрактного типа данных, 21
- Реализация языка, 15
- Регистр процессора, 8
- Редактор связей. *See* Компоновщик
- Реквизит документа, 7
- Рекомендация, 7
- Родительский узел, 25
- Руководство, 7
- Семантика, 12
- Семантический анализатор, 16
- Семиотика, 12
- Сигнатура подпрограммы, 18
- Синтаксис, 12
- Синтаксический анализатор, 16
- Синтактика, 12
- Система, 3
  - Данных, 19
  - Информационная, 4
  - Типов, 20
- Словарь, 23
  - Дважды упорядоченный, 23
  - Упорядоченный, 23
- Слово, 9
  - Машинное, 9
- Сложность
  - Временная, 27
  - Емкостная, 27
- Пространственная. *See*
  - Сложность:Емкостная
- Соглашение, 8
  - О вызове, 17
  - О стиле, 8
  - Об именовании, 8
- Сортировка, 27
- Состав, 3
- Спекулятивное выполнение, 9
- Специфика, 5
- Спецификация, 5
- Список, 21
- Ссылка, 20
- Стандарт, 7
- Стандартизация, 7
- Статический вектор. *See* Вектор:Статический
- Стек, 22
- Степень узла дерева, 25
- Структура, 3
  - Данных, 19
  - Компьютера, 8
- Структурная парадигма. *See* Парадигма программирования:структурная
- Суперскалярность, 9
- Тело подпрограммы, 17
- Техническое задание, 7
- ТЗ. *See* Техническое задание
- Тип данных, 19
  - Абстрактный, 21
  - Атомарный, 20
  - С управляемым временем жизни, 20
  - Составной, 20
- Тип переменной, 14
- Транслятор, 15
- Трансляция, 15
- Трасса поиска, 27
- Узел дерева, 24
- Указатель, 20
- Умолчание, 13
- Универсальный вектор. *See*
  - Вектор:Универсальный

- Фактический параметр. *See*  
    Параметр:Фактический
- Формальный параметр. *See*  
    Параметр:Формальный
- Формат данных, 19
- Фраза, 14
- Функция, 17
- Характеристика ПП
- Корректность, 6
  - Назначение, 5
  - Объём, 5
  - Открытость, 6
  - Переносимость, 6
  - Расширяемость, 6
  - Сложность, 6
  - Совместимость, 6
  - Универсальность, 6
  - Условия эксплуатации, 5
- Устойчивость, 6
- Целевая аудитория, 5
- Эффективность, 6
- Хвост очереди, 22
- Хэш-таблица, 24
- Хэш-функция, 24
- Эмерджентность, 3
- Язык
- Ассемблера, 10
  - Высокого уровня, 10
  - Декларативный, 11
  - Исходный, 15
  - Логический, 12
  - Машинный, 15
  - Программирования, 10
  - Функциональный, 12
  - Целевой, 15