Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

**И.Г. Самарина**
**А.Р. Бондарева**

# ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ
## Часть 3

*Утверждено Редакционно-издательским советом университета в качестве учебного пособия*

Магнитогорск
2015

УДК 004.4

**Рецензенты:**

Заместитель директора ООО «ЭПОЛ»
**О.Е. Савостьянов**

Кандидат педагогических наук,
доцент кафедры вычислительной техники и программирования,
ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова»
**Е.А. Ильина**

**Самарина И.Г., Бондарева А.Р.**
**Программирование и основы алгоритмизации. Часть 3** [Электронный ресурс]: учебное пособие / Ирина Геннадьевна Самарина, Альбина Робертовна Бондарева ; ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова». – Электрон. текстовые дан. (0,57 Мб). – Магнитогорск : ФГБОУ ВПО «МГТУ», 2015. – 1 электрон. опт. диск (CD-R). – Систем. требования : IBM PC, любой, более l GHz ; 512 Мб RAM ; 10 Мб HDD ; MS Windows XP и выше ; Adobe Reader 8.0 и выше ; CD/DVD-ROM дисковод ; мышь. – Загл. с титул. экрана.

Пособие написано в соответствии с учебной программой дисциплины «Программирование и основы алгоритмизации» и требованиями к обязательному минимуму содержания основной образовательной программы по направлению 220400 Управление в технических системах. Рекомендуется при изучении дисциплин: «Теория автоматического управления», «Моделирование систем» «Самонастраивающиеся системы», «Операционные системы реального времени», «Системы автоматизации и управления», «Автоматизированное управление в технических системах»; «Автоматизация технологических процессов и производств», «Оптимизация управления технологическими процессами металлургического производства», «Диагностика и надежность автоматизированных систем», «Базы данных в АСУ ТП» и выполнении курсовых проектов по дисциплинам непосредственно для студентов, обучающихся по направлению 220400 «Управление в технических системах» и других специальностей.

Изложены основы программирования, этапы разработки алгоритмов и написания программ. Включает краткие теоретические материалы, примеры.

УДК 004.4

© Самарина И.Г., Бондарева А.Р., 2015
© ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова», 2015

## СОДЕРЖАНИЕ

Введение ..... 4
1. Алгоритм и способы его записи ..... 6
1.1. Способы записи алгоритма ..... 13
1.2. Принципы проектирования алгоритмов ..... 18
1.2.1. Структурное проектирование ..... 18
1.2.2. Информационное моделирование ..... 22
1.2.3. Объектно-ориентированное проектирование ..... 22
1.3. Методы разработки алгоритмов ..... 25
1.3.1. Метод декомпозиции ..... 25
1.3.2. Динамическое программирование ..... 27
1.3.3. «Жадные» алгоритмы ..... 28
1.3.4. Поиск с возвратом ..... 28
1.3.5. Алгоритмы локального поиска ..... 29
2. Основные этапы алгоритмизации ..... 31
2.1. Постановка задачи ..... 31
2.2. Построение математической модели ..... 32
2.3. Разработка алгоритма решения задачи ..... 34
2.4. Программирование ..... 34
3. Этапы работ по созданию программных продуктов ..... 36
3.1. Составление технического задания на программирование ..... 36
3.2. Технический проект ..... 36
3.3. Рабочая документация (рабочий проект) ..... 36
3.4. Ввод в действие ..... 37
4. Жизненный цикл программных продуктов ..... 38
4.1. Маркетинг и спецификация программного продукта ..... 40
4.2. Проектирование структуры программного продукта ..... 41
4.3. Программирование, тестирование и отладка программ ..... 41
4.4. Документирование программного продукта ..... 44
4.5. Выход программного продукта на рынок программных средств ..... 46
4.6. Эксплуатация и сопровождение программного продукта ..... 47
4.7. Снятие программного продукта с продажи и отказ от сопровождения ..... 48
Вопросы экзаменационных билетов по разделу «Основные понятия программирования» ..... 49
Рекомендуемая литература ..... 51

# ВВЕДЕНИЕ

Решение задач с помощью компьютера предполагает использование различных программных средств, которые с разной степенью детализации процессов обработки преобразуют входные данные в выходные. Одна и та же задача может быть решена с использованием различных программных средств. Каждое из них имеет свой набор команд и реализует разные функции. Основой создания программ являются алгоритмизация и программирование.

В пособии речь будет идти о средствах и способах программирования, методах и этапах алгоритмизации задач. А так же рассмотрены основные этапы работ по созданию программного продукта.

В первом разделе пособия наряду с изложением общих понятий приводится принятая классификация вычислительных алгоритмов. Рассматриваются линейные, разветвляющиеся и циклические алгоритмы, а также приводятся примеры составления алгоритмов для различных прикладных задач.

Разработка любой программы начинается с построения алгоритма решения задачи. В пособии описываются особенности разработки алгоритмов для решения задач обработки информации на компьютере. Дана классификация современных принципов проектирования алгоритмов.

Проектирование алгоритмов и программ – наиболее ответственный этап жизненного цикла программных продуктов, определяющий, насколько создаваемая программа соответствует спецификациям и требованиям со стороны конечных пользователей. Затраты на создание, сопровождение и эксплуатацию программных продуктов, научно-технический уровень разработки, время морального устаревания и многое другое - все это также зависит от проектных решений. Методы проектирования алгоритмов и программ очень разнообразны и могут основываться на различных подходах.

Во втором разделе рассмотрены основные этапы алгоритмизации: от постановки задачи до программирования.

В третьем разделе этапы работ по созданию программных продуктов. Излагаются основные понятия и положения технологии разработки программных средств, позволяющие систематизировано представить вопросы нормирования и стандартизации жизненного цикла программной продукции, оценки технико-экономических показателей её разработки и эффективности соответствующих технологий проектирования.

И в четвертом разделе вопросы оценки трудоёмкости разработки программных средств. Дается определение жизненного цикла программных продуктов.

Современный подход к оценке трудоёмкости разработки программных средств состоит в учёте особенностей жизненного цикла программных средств на различных этапах и влияния технологических факторов не только на трудозатраты, но и на уровень качества, надёжность и экономические показатели программных средств.

Разработка программных средств является важнейшим элементом основных процессов жизненного цикла и в четвертом разделе представлены основные этапы жизненного цикла.

Быстрое развитие информационных технологий и расширение сферы их применения в последние годы привели к резкому росту разработок программного обеспечения.

Характеризуя общие тенденции в области информатизации, следует отметить высокую динамичность изменений технических и потребительских свойств информационных технологий и средств их реализации.

Опыт организации работ на всех фазах жизненного цикла программных средств показывает, что это сложная, трудоёмкая и длительная работа, требующая высокой

квалификации специалистов и новых подходов к проектированию на основе широкого использования методов программной индустрии, стандартизации и сертификации.

Чрезвычайно важными показателями для разработчиков и пользователей программной продукции являются трудоёмкость изготовления и сопровождения программного обеспечения, а также стоимостные показатели и уровень качества программной продукции. Поэтому этим вопросам в пособии уделено внимание.

# 1. АЛГОРИТМ И СПОСОБЫ ЕГО ЗАПИСИ

Алгоритм – одно из основных понятий современной математики и программирования. Слово происходит от имени узбекского математика IX в. Мухаммеда, уроженца Хорезма (по-арабски "Аль Хорезми"). Его работы по арифметике и алгебре были переведены на латинский язык в XII в. и оказали большое влияние на развитие математики в Европе. Сформулированные ученым правила выполнения четырех арифметических действий получили название "алгоризм", дальнейшая трансформация – "алгоритмус", "алгорифм" и "алгоритм".

Можно дать следующее определение алгоритма.

Алгоритм – некоторая конечная последовательность правил (предписаний), определяющая процесс преобразования исходных и промежуточных данных в результат решения задачи.

Разрабатываемый алгоритм должен обладать следующими свойствами:

1. массовостью, позволяющей решать не одну задачу, а целый класс задач, определяет возможность использования любых исходных данных из некоторого допустимого множества. Правило, сформулированное только для данного случая, которое не может быть использовано при других исходных данных, не является алгоритмом, например, таблица умножения не является алгоритмом, а правило умножения ”столбиком“ есть алгоритм;

2. детерминированностью, то есть однозначно определять выполняемые действия (промежуточные и окончательные результаты разных пользователей должны быть одинаковыми при одинаковых исходных данных);

3. дискретностью – разбиение процесса обработки информации на более простые этапы (шаги выполнения), при этом сложные функции обработки могут быть представлены через более простые последовательно выполняемые функции;

4. результативностью, позволяющей получить результат после конечного числа шагов.

По используемой структуре управления вычислительным процессом алгоритмы классифицируют следующим образом:

- линейной структуры;
- разветвляющейся структуры;
- циклической структуры;
- смешанной (комбинированной) структуры.

*Алгоритм линейной структуры* – алгоритм, в котором все действия выполняются последовательно друг за другом (см. рис. 1).

Каждый очередной блок линейной структуры обрабатывается после завершения предыдущего без дополнительных условий. Для изменения порядка обработки блоков выполняется редактирование их последовательности.

Алгоритм линейной структуры включает последовательное выполнение следующих этапов:

- ввод исходных данных в память ЭВМ;
- вычисление искомых величин по формулам;
- вывод результатов из памяти ЭВМ на информационный носитель.

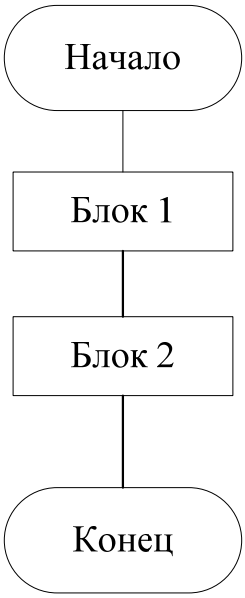



Рис. 1. Линейная структура алгоритма

***Пример 1.*** Составить алгоритм вычисления по формуле $S = X \cdot Y^2$. Решение представлено на рис. 2.

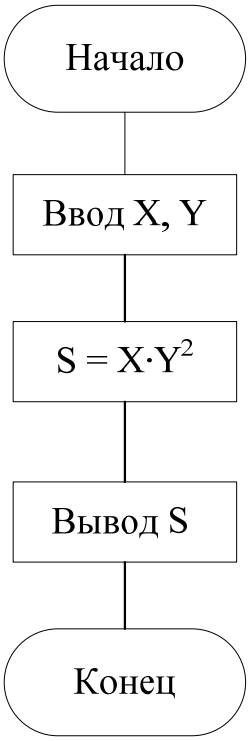



Рис. 2. Алгоритм вычисления $S = X \cdot Y^2$

Условные структуры алгоритмов часто называют ветвлениями, представлено на рис. 3. *Алгоритм разветвляющей структуры* – алгоритм, в котором в зависимости от выполнения некоторого логического условия вычислительный процесс должен идти по одной или другой ветви. Затем происходит выход на общее продолжение.

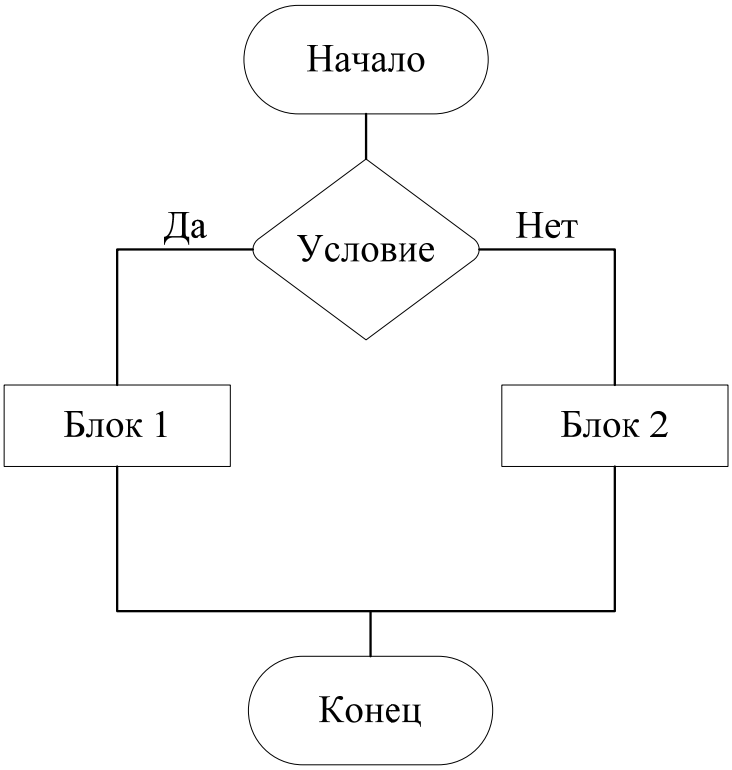



Рис. 3. Условная структура алгоритма

Данная структура содержит блок условие – альтернативные последовательности обработки данных в зависимости условия. Результатом выполнения условия является истина или ложь. В зависимости от него выполняется одна или другая ветвь. Обязательно выполняется одна из ветвей.

При программировании имеет место неполная форма ветвления, когда на ветви “нет” пусто.

***Пример 2***. Составить алгоритм решения для функции $Z(X) = X$ при $X > 0$ и $Z(X) = X^2$ при $X \leq 0$. Блок – схема разветвляющегося алгоритма показана на рис. 4

*Алгоритм циклической структуры* (рис. 5) – алгоритм, содержащий многократно выполняемые участки вычислительного процесса, называемые циклами. Если алгоритм содержит цикл, внутри которого размещен один или несколько других циклов, то такой алгоритм называется алгоритмом со структурой вложенных циклов.

Для организации цикла в блоке условия задается условие выполнения тела цикла. Если условие не выполняется, цикл прерывается и осуществляется выход.

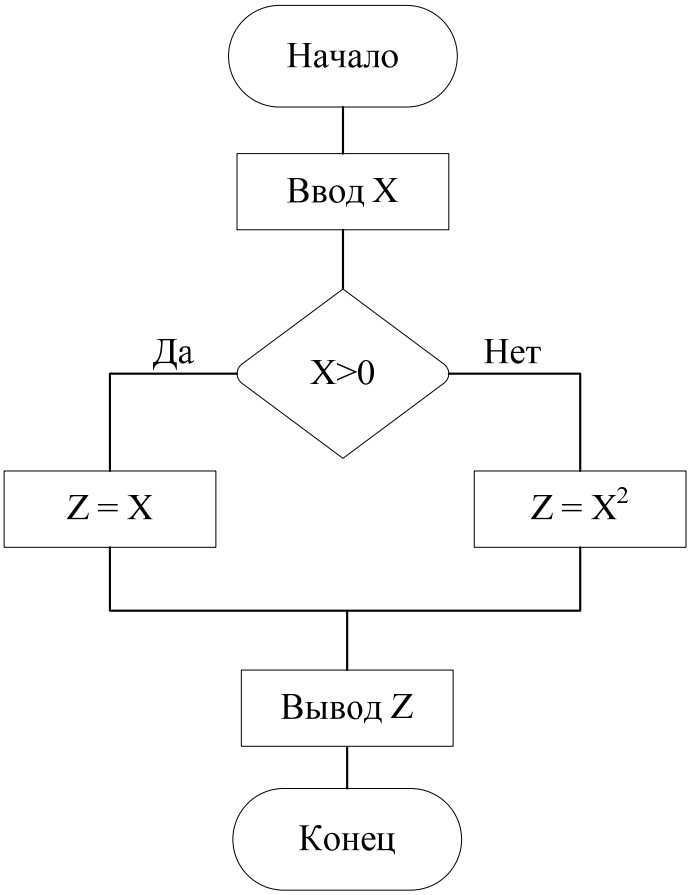



Рис. 4. Блок – схема разветвляющегося алгоритма

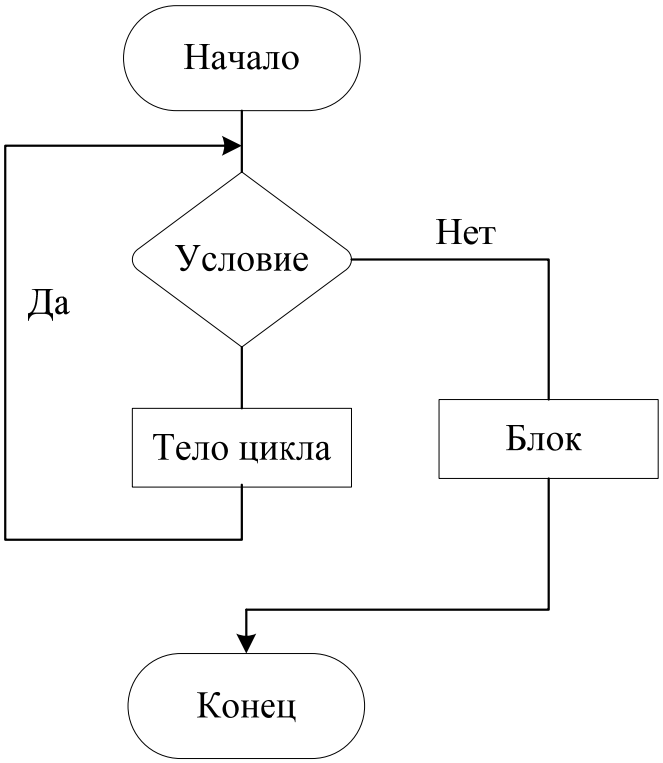



Рис. 5. Циклическая структура алгоритма

Цикл может повторяться заданное количество раз, если условие использует счетчик числа повторений, для которого задается начальное, конечное значение и шаг изменения.

Цикл может выполняться неизвестное количество раз, зависящее от того выполняется ли логическое условие.

***Пример 3.*** Для натурального N вычислить: $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\ldots+\frac{1}{\sin 1+\sin 2+\ldots+\sin N}$

Блок – схема циклического алгоритма показана на рис. 6

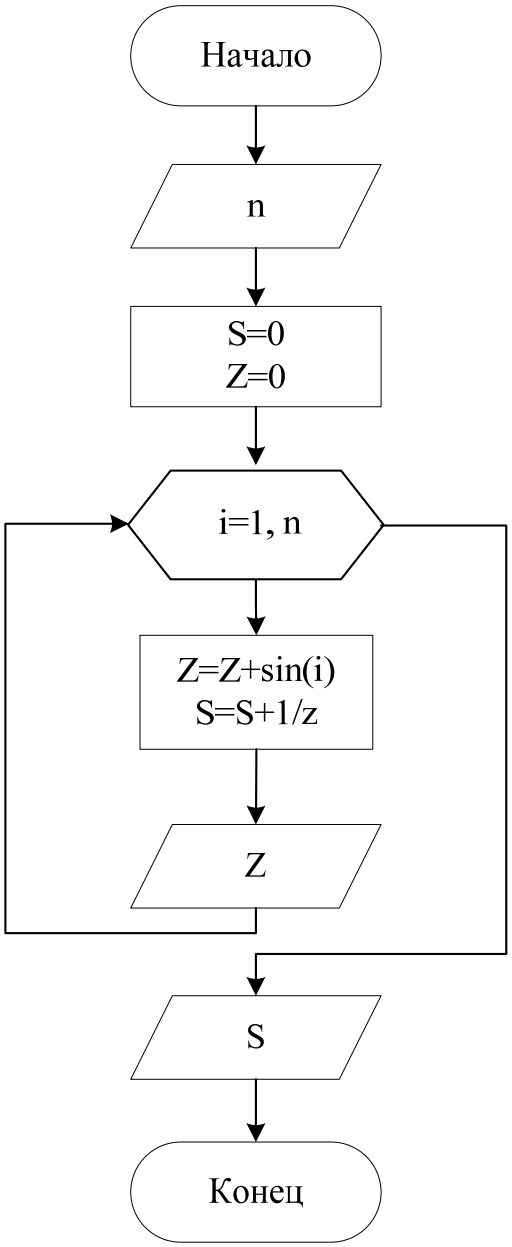



Рис. 6. Блок – схема циклического алгоритма

На практике чаще всего встречаются алгоритмы смешанного типа, у которых можно выделить участки (блоки), имеющие структуру линейного, ветвящегося или циклического типа. Алгоритм любой степени сложности можно построить с помощью блоков основного

базового набора, имеющих линейную, ветвящуюся или циклическую структуру. Каждая из этих структур имеет только один вход и только один выход, что позволяет соединять между собой в процессе разработки алгоритма любое количество элементов базовых структур в любой последовательности.

Порядок выполнения алгоритма:

– действия в алгоритме выполняются в порядке их записи;

– нельзя менять местами никакие два действия алгоритма;

– нельзя не закончив одного действия переходить к следующему.

При организации циклов с известным числом повторений присутствуют стандартные элементы, сопровождающие любой цикл:

– подготовка первого выполнения цикла (присвоение счетчику цикла начального значения);

– тела цикла, которое образуют блоки, выполняемые многократно;

– изменение значения счетчика циклов и сравнение его с конечным значением.

Блок-схемы циклических алгоритмов существенно отличаются структурами повторения "*повторять ДО*" (повторять до выполнения условия окончания цикла) или "*повторять ПОКА*" (повторять пока выполняются условия продолжения циклического процесса). В первом варианте проверка условий окончания циклических вычислений осуществляется в конце цикла, а во втором – в начале цикла. Цикл "*повторять ДО*" выполняется, по крайней мере, один раз, а цикл "*повторять ПОКА*" может сразу привести к выходу из цикла.

Примером циклов с неизвестным числом повторений являются итерационные вычислительные процессы. В них решение задачи реализуется путем последовательного приближения к искомому результату. Процесс является циклическим, поскольку заключается в многократных вычислениях. Начальное приближение выбирается заранее или задается по определенным правилам. Заканчивается итерационное вычисление при выполнении условия: $Y_i - Y_{i-1} \leq \varepsilon$, где $\varepsilon$ – допустимая ошибка вычисления.

То есть, алгоритм служит основой для создания программ. Программа для ЭВМ представляет собой описание алгоритма и данных на некотором языке программирования.

*Язык программирования* – формализованный язык для описания алгоритма решения задачи на компьютере.

Языки программирования делятся на две группы:

1. Машинно-зависимые языки, которые можно применять на одной ЭВМ или на ограниченном подмножестве машин с одинаковой архитектурой.

2. Машинно-независимые языки - их можно использовать на любой ЭВМ. Языки этой группы называют универсальными языками.

*Машинно-зависимые языки* в зависимости от их близости к машинным языкам, делятся на три группы:

– машинные языки (языки нулевого уровня);

– ассемблерные языки (языки первого уровня или языки типа 1:1, последнее означает, что одна ассемблерная команда после трансляции порождает ровно одну машинную команду);

– макроассемблеры (языки второго уровня или языки типа 1:n).

Аналогично, *машинно-независимые языки* включают следующие группы языков:

– процедурно – ориентированные языки: С, Паскаль, ФОРТРАН, БЭЙСИК и др. Процедурные языки требуют детальной разработки алгоритма решения и, по существу, являются языками для записи алгоритмов решения задач;

– проблемно – ориентированные языки или языки типа "заполни бланк". Это языки описания задач, специализированные языки (ЛИСП и др.). Используя подобный язык программирования, пользователь сообщает только, какую задачу надо решить и с какими данными.

Как решить задачу - "знает" язык. В качестве примера проблемного языка можно назвать язык ПРОСПО, разработанный фирмой IBM для программирования систем управления производственными процессами.

– универсальные языки: ПЛ/1, АЛГОЛ-68, Ада и др. При создании универсальных языков в их состав включили все лучшее, что имелось на момент создания в процедурных языках.

– объектно-ориентированные (C++, Visual Basic, Java и др.)

Любой язык полностью определяется заданием алфавита (словаря исходных символов), точным описанием их синтаксиса (грамматики) и семантики (смысла).

*Алфавит языка* – набор основных символов (*литер*), используемых для записи алгоритма. Некоторые литеры алфавита являются составными, изображаются двумя или тремя символами, но рассматриваются как неделимые (например, "+=" или "»=").

*Семантика* определяет смысл предложений (*операторов*), записанных на языке, как каждого в отдельности, так и их совокупности.

При написании программ используются символы языка программирования, составляющие его *алфавит.* Набор символов зависит от среды выполнения.

На ЭВМ широко используется символьный набор ISO 6461983, называемый кодом ASCII (American Standard Code for Information Interchange – американский стандартный код обмена информаций). Он содержит латинские буквы, арабские цифры, специальные и управляющие символы.

Эти символы в своем большинстве входят в состав *алфавита языка* ***C*++.** Каждый символ кодируется 7-ми битным значением. Для представления *кириллических символов и*спользуется 8-ми битный *расширенный* ASCII-код, в котором единичное значение старшего бита, говорит об использовании дополнительного символьного набора.

С помощью языка программирования получается не готовая для выполнения программа, а только ее текст, описывающий разработанный ранее алгоритм. Чтобы получить работающую программу, необходимо использовать трансляторы.

*Трансляторы* – программы, переводящие исходный текст на язык машинных кодов (трансляция) и настраивающие адресные константы для работы программы с библиотечными функциями (компоновка).

По способу работы трансляторы подразделяются на компиляторы и интерпретаторы. Компилятор запускается после окончания ввода программы и просматривает текст программы целиком в поисках синтаксических ошибок. В случае обнаружения ошибок необходимо выполнить отладку программы для их ликвидации. Если в процессе компиляции программы ошибки не обнаружены, то генерируется машинный код (готовая для выполнения программа).

В отличие от компилятора, *интерпретатор* просматривают текст программы последовательно на этапе ввода. Проверка отдельно каждого оператора на наличие синтаксических ошибок осуществляется его формальным выполнением. Интерпретатор переходит к следующему оператору только после успешного выполнения предыдущего.

## 1.1. Способы записи алгоритма

Один и тот же алгоритм может быть записан (описан) различными способами. Наибольшее распространение в практике программирования получили:

*1. Словесный (текстуальный) способ записи алгоритма*

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Словесный способ не имеет широкого распространения, так как такие описания:

- строго не формализуемы;
- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

Имеется следующий алгоритм, записанный в текстуальной форме:

1. Начало алгоритма
2. Выполнить некоторое действие (оператор) S1
3. Если выполнено условие "Условие 1",
то выполнить операторы S2, S3
и перейти к п. 4 иначе – перейти к п. 3.1
3.1. Пока выполняется условие "Условие 2",
выполнять п. 3.2 и 3.3
иначе – перейти к п. 4
3.2. Если выполнено условие "Условие 3",
то выполнить оператор S4,
иначе – выполнить оператор S5
3.3. Выполнить оператор S6
4. Пока выполняется условие "Условие 4",
выполнять оператор S7
иначе – перейти к п. 5
5. Выполнить оператор S8
6. Конец алгоритма

*2. Запись алгоритма с помощью схемы – разновидность визуального формализма (ГОСТ 19.701 - 90, совместим с международным стандартом).*

Запись того же самого алгоритма в виде схемы показано на рис. 8.

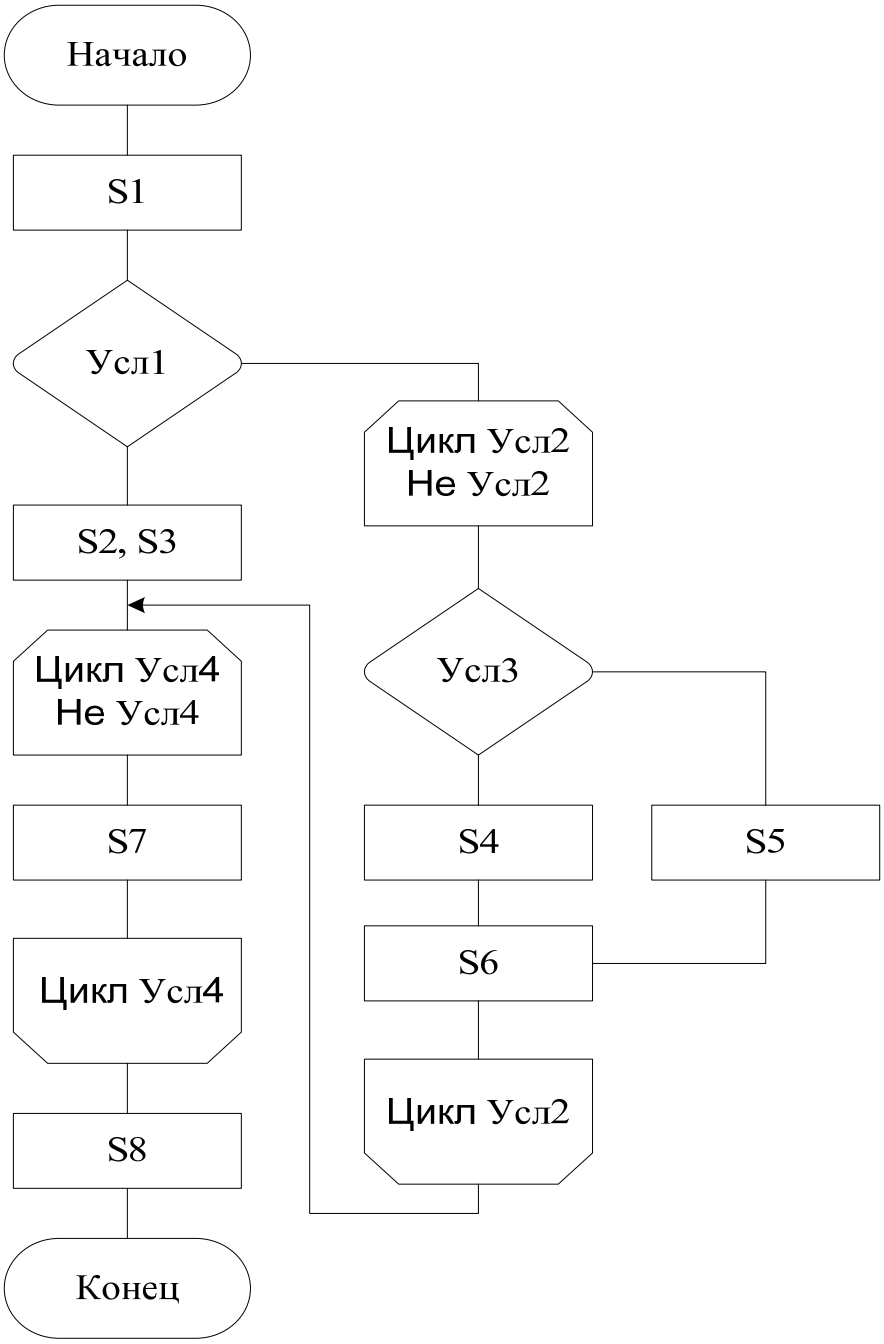



Рис. 8. Блок-схема алгоритма

Блок-схема – это графический способ представления алгоритма, каждое действие при этом осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма. Порядок выполнения действий указывается стрелками. В таблице 1 приведены наиболее часто употребляемые блоки.

Таблица 1

Наиболее часто употребляемые блоки

| № | Символ | Наименование | Содержание |
|---|---|---|---|
| 1 | | Начало (конец) | Начало или конец алгоритма, вход или выход в программу |
| 2 | | Блок ввода-вывода данных | 1. Общие обозначения ввода (вывода) данных (вне зависимости от физического носителя)<br>2. Вывод данных, носителем которых является документ |
| 3 | | Блок вычислений | Вычислительные действия или последовательность действий |
| 4 | | Логический блок | Выбор направления выполнения алгоритма в зависимости от некоторого условия |
| 5 | | Процесс пользователя (подпрограмма) | Вычисление по стандартной программе (подпрограмме) |
| 6 | | Документ | Вывод данных на бумажный носитель |
| 7 | | Магнитный диск | Ввод-вывод данных носителем которых является магнитный диск |
| 8 | | Блок модификации | Функция выполняет действия, изменяющие пункты (например, заголовок цикла) алгоритма |
| 9 | | Соединитель | Указание связи прерванными линиями между потоками информации в переделах одного листа |
| 10 | | Межстраничные соединения | Указание связи между потоками информации на разных листах |

В случае простой схемы сравнительно несложно обеспечить ее бесспорную наглядность. Но по мере роста сложности отображаемого фрагмента алгоритма (программы), его логическая структура становится тяжело читаемой. Поэтому практика использования схем алгоритмов считается устаревшей. Там, где стандарты организации требуют наличия схем алгоритмов, они рисуются после написания программы.

Еще две разновидность визуального формализма это с использованием *диаграммы Нэсси-Шнейдермана* и *запись алгоритма с использованием Р-схемы.*

Диаграмма Нэсси-Шнейдермана была предложена в сочетании со структурным программированием как средство борьбы с проблемой присущей схемам алгоритмов. Эта диаграмма, бесспорно, заменяет одномерное представление вложенных операторов двумерным, представлена на рис. 9. Но по мере роста сложности программного кода появляются проблемы отображения, так как элементы диаграммы быстро становятся все меньше и меньше. На рис. 9 приведена диаграмма Нэсси-Шнейдермана для того же самого алгоритма, что и ранее.

Р-технология программирования разработана в Институте Кибернетики АН УССР. Согласно Р-технологии программа должна быть представлена в форме нагруженного по дугам структурного графа (Р-схемы). Р-схема состоит из подграфов, каждый из которых имеет один вход и один выход. Вершины графа называются состояниями Р-схемы. Переходы из одного состояния в другое представлены помеченными дугами, причем каждая дуга может быть помечена условием перехода и действием, выполняемым в процессе выполнения перехода. На рис. 10 приведена Р-схема для того же самого алгоритма.

*3. Запись алгоритма с помощью псевдокода*

Данный способ записи алгоритма получил широкое распространение.

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.

Этот способ записи является промежуточным и используется перед записью алгоритма в терминах выбранного языка программирования. Псевдокод представляет собой удобный для практики промежуточный язык. Это и не естественный язык, и не язык программирования, а их симбиоз. Псевдокод похож на язык программирования тем, что может использовать его некоторые инструкции, но, с другой стороны, допускает и словесную, и формульную записи там, где сразу сложно воспользоваться языком программирования.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

*4. Программный способ записи (тексты на языках программирования)*

Язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке – программой для компьютера.

Эти способы записи алгоритмов не исключают друг друга, а используются последовательно, на различных этапах решения задачи.

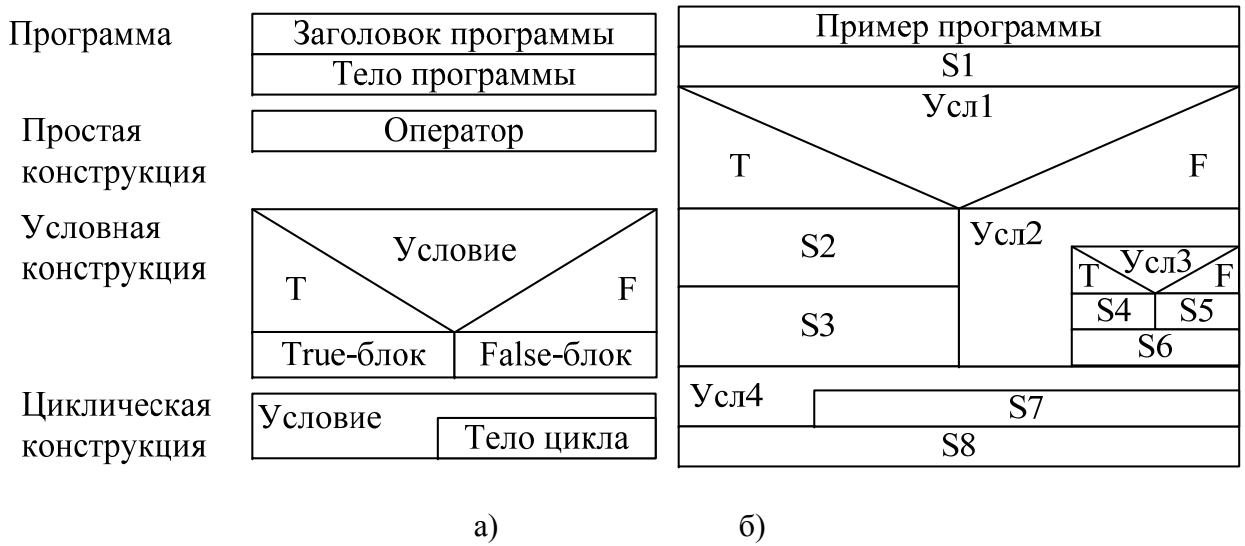



Рис. 9. Диаграмма Нэсси-Шнейдермана

а) графические элементы; б) пример использования

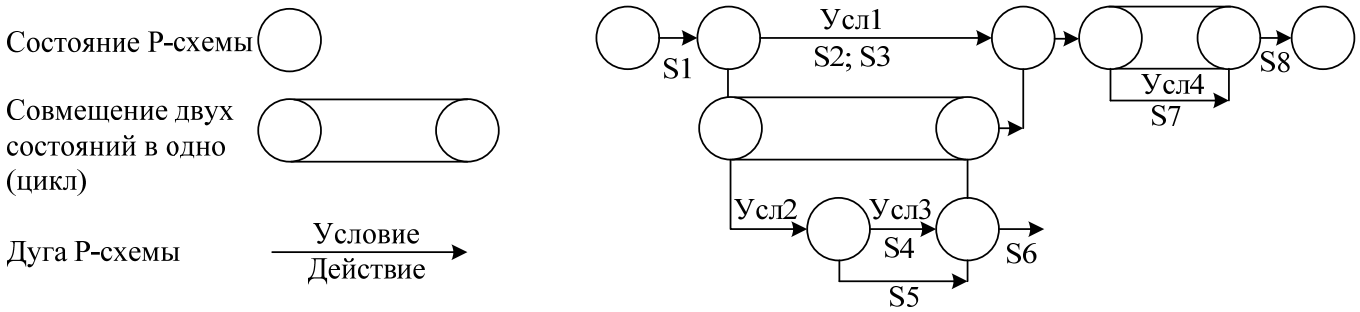



Рис. 10. Р-схема

а) графические элементы; б) пример использования

## 1.2. Принципы проектирования алгоритмов

Создание программных продуктов является трудоемким процессом, основанным на определенной технологии и инструментарии его разработки. Проектирование алгоритмов и программ – наиболее ответственный этап жизненного цикла программных продуктов, определяющий, насколько создаваемая программа соответствует спецификациям и требованиям со стороны конечных пользователей. Затраты на создание, сопровождение и эксплуатацию программных продуктов, научно-технический уровень разработки, время морального устаревания и многое другое - все это также зависит от проектных решений.

Методы проектирования алгоритмов и программ очень разнообразны, их можно классифицировать по различным признакам, важнейшими из которых являются:

- степень автоматизации выполнения проектных работ;
- принятая методология процесса разработки.

По степени автоматизации процесса проектирования алгоритмов и программ можно выделить:

- методы традиционного (неавтоматизированного) проектирования;
- методы автоматизированного проектирования (CASE- технология и ее элементы).

Неавтоматизированное проектирование алгоритмов и программ преимущественно используется при разработке небольших по трудоемкости и структурной сложности программных продуктов, не требующих участия большого числа разработчиков. Трудоемкость разрабатываемых программных продуктов, как правило, небольшая, а сами программные продукты имеют преимущественно прикладной характер.

Автоматизированное проектирование алгоритмов и программ возникло в связи с необходимостью уменьшения затрат на проектные работы, сокращения сроков их выполнения, создания типовых "заготовок" алгоритмов и программ, многократно тиражируемых для различных разработок, координацией работ большого коллектива разработчиков, стандартизацией алгоритмов и программ.

Автоматизация проектирования может охватывать все или отдельные этапы жизненного цикла программного продукта, при этом работы этапов могут быть изолированы друг от друга, либо составлять единый комплекс, выполняемый последовательно во времени. Автоматизированное проектирование алгоритмов и программ под силу лишь крупным фирмам, специализирующимся на разработке определенного класса программных продуктов, занимающих устойчивое положение на рынке программных средств.

Среди современных принципов проектирования алгоритмов и программ наиболее распространенными являются:

- структурное проектирование программных продуктов;
- объектно-ориентированное проектирование программных продуктов;
- информационное моделирование предметной области.

До появления этих подходов для решения задач на компьютере использовалась процедурная (алгоритмическая) организация структуры программ.

### *1.2.1. Структурное проектирование*

В 1965 г. профессор Эйндховенского университета Дейкстра (Нидерланды) начал пропагандировать стиль программирования, получивший название "программирование без оператора *goto* (безусловного перехода)", первоначально принятый большинством

программистов негативно. Однако, в течение нескольких последующих лет этот же стиль, получивший название *структурного программирования,* нашел широкое применение.

Основными управляющими конструкциями структурного программирования являются:

– *следование* – если в записи алгоритма (программы) подряд написаны несколько действий (операторов) друг за другом, то они будут выполняться последовательно в таком же порядке.

– *если–то–иначе* – условная конструкция, определяющая разветвление в порядке выполнения действий (операторов). Дословный перевод этой конструкции if-then-else.

– *циклы*, в структурном программировании предусмотрены циклические конструкции трех видов:

1. Цикл с предусловием *while (*дословный перевод с английского этой конструкции *пока делай):* пока истинно некоторое условие, делай то-то и то-то.

2. Цикл с постусловием *do-while (повторяй пока): о*тличается от предыдущего цикла тем, что тело цикла повторяется не менее одного раза.

3. Цикл с заранее заданным числом повторений *(for).*

Этих трех элементарных конструкций, называемых базовыми конструкциями структурного программирования, достаточно, чтобы управлять порядком выполнения действий в любом алгоритме. Каждая из названных конструкций имеет только один вход и один выход, что делает их использование очень удобным.

Наряду с методологией структурного программирования, хороший стиль программирования рекомендует также обязательное использование методологии *модульного программирования*. Модульное программирование предполагает последовательную декомпозицию (разбиение) исходной задачи на функционально законченные подзадачи, оформленные в виде отдельных модулей, которые в языках C/C++ называются функциями.

Модульное программирование, получившее также название *нисходящего программирования*, для сложных программных проектов может носить иерархический характер, т.е. полученные вначале программные модули, в свою очередь, при необходимости, также декомпозируются, с тем, чтобы достичь указанных выше показателей, соответствующих хорошему стилю программирования.

Модульное программирование основано на понятии модуля - логически взаимосвязанной совокупности функциональных элементов, оформленных в виде отдельных программных модулей.

Модуль должен обладать следующими свойствами:

– один вход и один выход - на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, то есть реализуется стандартный - вход-процесс-выход;

– функциональная завершенность программного модуля - модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;

– логическая независимость - результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;

– слабые информационные связи с другими программными модулями - обмен информацией между модулями должен быть по возможности минимизирован;

– обозримый по размеру и сложности программный элемент.

Таким образом, модули содержат определение доступных для обработки данных, операции обработки данных, схемы взаимосвязи с другими модулями.

Каждый модуль состоит из спецификации и тела. Спецификации определяют правила использования модуля, а тело - способ реализации процесса обработки.

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования. Сначала определяется состав и подчиненность функций, а затем - набор программных модулей, реализующих эти функции.

Однотипные функции реализуются одними и теми же модулями. Функция верхнего уровня обеспечиваются главным модулем, который управляет выполнением нижестоящих функций, которым соответствуют подчиненные модули.

При определении набора модулей, реализующих функции конкретного алгоритма, необходимо учитывать следующее:

– каждый модуль вызывается на выполнение вышестоящим модулем и, закончив работу, возвращает управление вызвавшему его модулю;

– принятие основных решений в алгоритме выносится на максимально высокий по иерархии уровень;

– для использования одной и той же функции в разных местах алгоритма создается один модуль, который вызывается на выполнение по мере необходимости.

В результате дальнейшей детализации алгоритма создается функционально-модульная схема (ФМС) алгоритма приложения, которая является основой для программирования.

Состав и вид программных модулей, их назначение и характер использования в программе в значительной степени определяется инструментальными средствами.

Алгоритмы большой сложности обычно представляются с помощью схем двух видов:

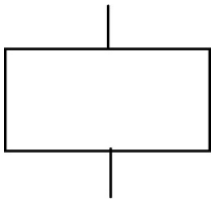
– обобщенной схемы алгоритма – раскрывающей общий принцип функционирования алгоритма и основные логические связи между отдельными модулями на уровне типовых процессов обработки информации (ввод и редактирование данных, вычисления, печать результатов и т.п.);

– детальной схемы алгоритма – представляющей содержание каждого элемента обобщенной схемы с использованием управляющих структур в блок-схемах алгоритма, псевдокода либо алгоритмических языков высокого уровня.

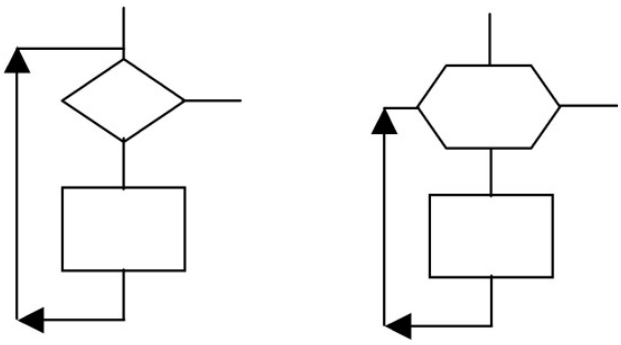
Наиболее часто детально проработанные алгоритмы изображаются в виде блок-схем с использованием графических элементов согласно требованиям структурного программирования.

При работе над модулем можно применить принцип структурного программирования. Его цель – повышение читабельности и ясности алгоритма (и программы), более высокой производительности программистов и упрощение отладки. В соответствии с этим принципом для построения любого алгоритма (программы) требуются три типовых блока:

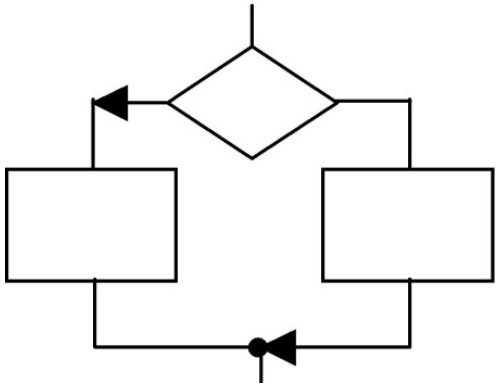
1. *функциональный блок*, используется для представления линейных алгоритмов, описывается языком графических символов, следующим образом:



2. *циклический блок*, используется для представления циклических алгоритмов. Описывается языком графических символов одним из двух способов:



3. *конструкция принятия двоичного решения*, применяется для представления разветвляющих алгоритмов, описывается языком графических символов, следующим образом:



В зависимости от объекта структурирования, различают:

– функционально-ориентированные методы - последовательное разложение задачи или целостной проблемы на отдельные, достаточно простые составляющие, обладающие функциональной определенностью;

– методы структурирования данных.

Для функционально-ориентированных методов характерна ориентация, в первую очередь, на заданные функции обработки данных, в соответствии с которыми определяется состав и логика (алгоритмы) работы отдельных компонентов программного продукта.

Основной упор в структурном подходе делается на моделировании процессов обработки данных. Для методов структурирования данных осуществляется анализ, структурирование и создание моделей данных, применительно к которым устанавливается необходимый состав функций и процедур обработки. Программные продукты тесно связаны со структурой обрабатываемых данных, изменение которой отражается на логике обработки (алгоритмах) и обязательно требует перепроектирования программного продукта.

Основной графический инструментарий структурного подхода, который также нашел применение в современных CASE-технологиях, это:

– диаграммы потоков данных (информационно-технологические схемы) - показывают процессы и информационные потоки между ними с учетом "событий", инициирующих процессы обработки;

– интегрированная структура данных предметной области (инфологическая модель, ER-диаграммы);

– диаграммы декомпозиции - структура и декомпозиция целей, функций управления, приложений;

– структурные схемы - архитектура программного продукта в виде иерархии взаимосвязанных программных модулей с идентификацией связей между ними, детальная логика обработки данных программных модулей (блок-схемы).

Для полного представления о программном продукте необходима также текстовая информация описательного характера.

### *1.2.2. Информационное моделирование*

Информационные модели и структуры данных имеют большое значение для информационного моделирования предметной области. Роль данных при проектировании алгоритмов и программ в информационных моделях является определяющей. Составляющие данного подхода:

– информационный анализ предметных областей (бизнес-областей);

– информационное моделирование - построение комплекса взаимосвязанных моделей данных;

– системное проектирование функций обработки данных;

– детальное конструирование процедур обработки данных.

Первоначально строятся информационные модели различных уровней представления:

– информационно-логическая модель - не зависящая от средств программной реализации хранения и обработки данных, отражающая интегрированные структуры данных предметной области;

– даталогические модели, ориентированные на среду хранения и обработки данных.

Даталогические модели имеют логический и физический уровень представления. Физический уровень соответствует организации хранения данных в памяти компьютера. Логический уровень данных применительно к СУБД реализован в виде:

– концептуальной модели базы данных - интегрированные структуры данных под управлением СУБД;

– внешних моделей данных - подмножество структур данных для реализации приложений.

Средствами структур данных моделируются функции предметной области, прослеживается взаимосвязь функций обработки, уточняется состав входной и выходной информации, логика преобразования входных структур данных в выходные. Алгоритм обработки данных можно представить как совокупность процедур преобразований структур данных.

Выбор средств реализации базы данных определяет вид даталогических моделей и, следовательно, алгоритм преобразования данных. В большинстве случаев используется реляционное представление данных и соответствующие реляционные языки для программирования (манипулирования) обработки данных СУБД и реализации алгоритмов обработки.

### *1.2.3. Объектно-ориентированное проектирование*

Объектно-ориентированный подход к проектированию программных продуктов основан на:

– выделении классов объектов;

– установлении характерных свойств объектов и методов их обработки;

– создании иерархии классов, наследовании свойств объектов и методов их обработки.

Каждый объект объединяет в себе как данные, так и программу обработки этих данных и относится к определенному классу. С помощью класса одну и ту же программу можно использовать для относящихся к нему различных объектов.

Объектный подход при разработке алгоритмов и программ предполагает:

– объектно-ориентированный анализ предметной области и выделение объектов, определение свойств и методов обработки объектов, установление их взаимосвязей;

– объектно-ориентированное проектирование соединяет процесс объектной декомпозиции и представления с использованием моделей данных проектируемой системы на логическом и физическом уровнях, в статике и динамике.

Для проектирования программных продуктов разработаны объектно-ориентированные технологии, которые включают в себя специализированные языки программирования и инструментальные средства разработки пользовательского интерфейса.

Традиционные подходы к разработке программных продуктов всегда подчеркивали различия между данными и процессами их обработки. Так, технологии, ориентированные на информационное моделирование, сначала специфицируют данные, а затем описывают процессы, использующие эти данные. Технологии структурного подхода ориентированы, в первую очередь, на процессы обработки данных с последующим установлением необходимых для этого данных и организации информационных потоков между связанными процессами. Компьютерные программы отдельных программных модулей обеспечивают обработку входных данных, сами же модули не содержат данных.

Объектно-ориентированная технология разработки программных продуктов и данные, и процессы объединяет в логические сущности - объекты, которые имеют способность наследовать характеристики (методы и данные) одного или более объектов, тем самым обеспечивая повторное использование программного кода. Это приводит к значительному уменьшению затрат на создание программных продуктов, повышает эффективность жизненного цикла программных продуктов (сокращается длительность фазы разработки). При выполнении программы объекту посылается сообщение, которое инициирует обработку данных объекта.

Объектно-ориентированный подход использует следующие базовые понятия: объект, свойство объекта, метод обработки, событие, класс объектов.

*Объект* – совокупность свойств (параметров) определенных сущностей и методов их обработки (программных средств). Содержит инструкции (программный код), определяющие действия, которые может выполнять сам объект, и обрабатываемые данные.

*Свойство* – характеристика объекта, его параметр. Все объекты наделены определенными свойствами, которые в совокупности выделяют объект из множества других объектов.

*Метод обработки* – программа действий над объектом или его свойствами. Рассматривается как программа, связанная с определенным объектом, осуществляющая преобразование свойств, изменяющая поведение объекта.

Объект может обладать набором заранее определенных встроенных методов обработки, либо созданных пользователем или заимствованных в стандартных библиотеках, которые выполняются при наступлении заранее определенных событий.

*Событие* – изменение состояния объекта происходит в момент совершения события, которое инициирует действие программы. Внешние события генерируются пользователем (например, клавиатурный ввод или нажатие кнопки мыши, выбор пункта меню, запуск макроса). Внутренние события генерируются системой.

Объекты могут объединяться в *классы*. Это совокупность объектов, характеризующаяся общностью применяемых методов обработки или свойств. Один объект может выступать объединением вложенных в него по иерархии других объектов.

Существуют различные объектно-ориентированные технологии и методики проектирования программных продуктов, которые должны обеспечить выполнение важнейших принципов объектного подхода:

- инкапсуляция;
- полиморфизм;
- абстракция;
- наследование.

*Инкапсуляция* (дословно – «содержание в оболочке») представляет собой объединение в рамках объекта, как единого целого, данных и функций, обрабатывающих эти данные. В совокупности они отражают свойства объекта (в C++ данные класса и объекта называются элементом данных или полями, а функции – методами или элементами-функциями).

*Или инкапсуляция* (encapsulation) – это, механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования.

Цель инкапсуляции – автономность модулей. При задании класса инкапсуляция обеспечивается заданием *спецификаторов доступа*.

*Полиморфизм* (polymorphism) – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному подходу, является использование одного имени для задания общих для класса действий. Выполнение каждого действия будет определяться типом данных.

Например, в языке C имеются функции: abs() – целое, fabs() – плавающее, labs() – длинное, а в C++ достаточно одной функции abs().

Тип данных, используемый при вызове функции, определяет, какая версия функции выполняется. Даная операция называется перегрузкой функций (function overloading).

Полиморфизм может применяться также и к операторам – перегрузка операторов (operator overloading).

Концепция полиморфизма – «один интерфейс, множество решений».

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному – интерфейс, класс. Общность имеет внешнее и внутреннее выражение. Внешне общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов). Внутренняя общность есть одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (функций, операций).

*Абстракция* – это придание объекту характеристик, которые чётко определяют его концептуальные границы, отличая от всех других объектов.

Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов.

Абстракция является основой объектно-ориентированного программирования и позволяет работать с объектами, не вдаваясь в особенности их реализации.

*Наследование* (inheritance) – один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и

абстракцией). Это процесс, с помощью которого объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним свои характерные только для него свойства.

Концепция иерархии классов – управление большими потоками информации.

Например, есть класс B, обладающий всеми свойствами класса A и имеющий свои дополнительные свойства, то класс A называется базовым (родительским) классом, а B – производным (наследником) классом.

В процессе объектно-ориентированного анализа:

- осуществляется идентификация объектов и их свойств;
- устанавливается перечень операций (методов обработки), выполняемых над каждым объектом, в зависимости от его состояния (событий);
- определяются связи между объектами для образования классов;
- устанавливаются требования к интерфейсу с объектами.

Выделены этапы объектно-ориентированного проектирования:

- разработка структуры классов, описывающей связь между классами и объектами;
- разработка диаграмм объектов, показывающих взаимосвязи с другими объектами;
- разработка внутренней структуры программного продукта.

## 1.3. Методы разработки алгоритмов

В настоящее время специалисты по вычислительной технике разработали ряд методов, которые позволяют получать эффективные алгоритмы решения больших классов задач. Некоторые из наиболее важных методов, такие как "разделяй и властвуй" (декомпозиции), динамическое программирование, "жадные" методы, поиск с возвратом и локальный поиск, представлены далее.

Однако, существуют задачи, например NP-полные (non-deterministic polynomial) задачи, для которых эти и любые другие известные методы не обеспечивают эффективных решений. Когда встречается подобная задача, нередко бывает полезно определить, обладают ли входные данные для этой задачи какими-либо особыми характеристиками, которыми можно воспользоваться при попытках найти требуемое решение, и можно ли воспользоваться каким-либо приближенным решением (если такое решение легко получить) вместо точного решения, получить которое бывает очень трудно.

### *1.3.1. Метод декомпозиции*

*Метод декомпозиции* (или метод разбиения, или метод «разделяй и властвуй») является одним из самых распространенных методов проектирования алгоритмов. Он предполагает такую декомпозицию (разбиение) задачи размера n на более мелкие подзадачи, так что на основе решения этих более мелких подзадач можно легко получить решение исходной задачи. Как правило, алгоритмы, разработанные этим методом, отличаются легкостью разработки и высокой эффективностью.

Многие полезные алгоритмы имеют рекурсивную структуру: для решения данной задачи они рекурсивно вызывают сами себя один или несколько раз, чтобы решить вспомогательную задачу, имеющую непосредственное отношение к поставленной задаче. Такие алгоритмы зачастую разрабатываются с помощью метода декомпозиции, или разбиения: сложная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются

рекурсивным методом, после чего полученные решения комбинируются с целью получить решение исходной задачи.

Парадигма, лежащая в основе метода декомпозиции "разделяй и властвуй", на каждом уровне рекурсии включает в себя три этапа.

– разделение задачи на несколько подзадач.

– покорение – рекурсивное решение этих подзадач, когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.

– комбинирование решения исходной задачи из решений вспомогательных задач.

Пример метода декомпозиции является *алгоритм сортировки слиянием* (merge sort). Его работу можно описать таким образом:

– разделение: сортируемая последовательность, состоящая из n элементов, разбивается на две меньшие последовательности, каждая из которых содержит n/2 элементов

– покорение: сортировка обеих вспомогательных последовательностей методом слияния

– комбинирование: слияние двух отсортированных последовательностей для получения окончательного результата.

Проиллюстрируем этот метод на примере «Ханойские башни» (рис. 11). Имеются три стержня A, B и C. Вначале на стержень A нанизаны несколько дисков: диск наибольшею диаметра находится внизу, а выше – диски последовательно уменьшающеюся диаметра. Цель головоломки – перемещать диски (по одному) со стержня на стержень так, чтобы диск большею диаметра никогда не размещался выше диска меньшею диаметра и чтобы в конце концов все диски оказались нанизанными на стержень B. Стержень C можно использовать для временною хранения дисков.

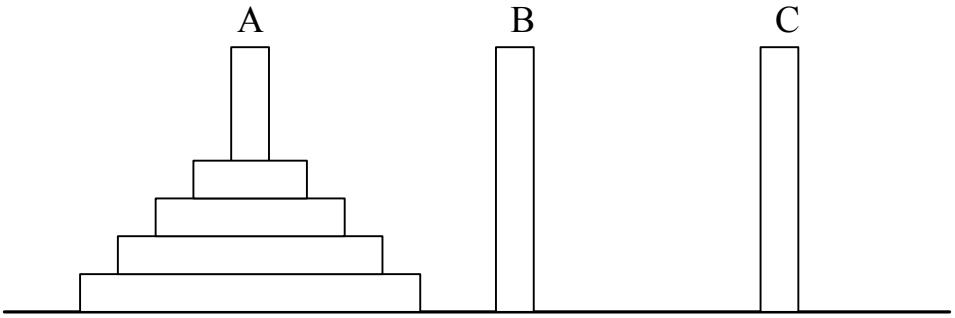



Рис. 11. Головоломка «Ханойские башни»

Задачу перемещения n наименьших дисков со стержня A на стержень B можно представить себе состоящей из двух подзадач размера n – 1. Сначала нужно переместить n – 1наименьших дисков со стержня A на стержень C, оставив на стержне A n-й наибольший диск. Затем этот диск нужно переместить с A на B. Потом следует переместить n – 1 дисков со стержня C на стержень B. Это перемещение n – 1 дисков выполняется путем рекурсивною применения указанною метода. Поскольку диски, участвующие в перемещениях, по размеру меньше тех, которые в перемещении не участвуют, не нужно задумываться над тем, что находится под перемещаемыми дисками на стержнях A, B или C. Хотя фактическое перемещение отдельных дисков не столь очевидно, а моделирование вручную выполнить непросто из-за образования стеков рекурсивных вызовов, с концептуальной точки зрения этот алгоритм все же довольно прост для понимания и доказательства его правильности (а если говорить о быстроте разработки, то ему вообще нет равных). Именно легкость

разработки алгоритмов по методу декомпозиции обусловила огромную популярность этого метода; к тому же во многих случаях эти алгоритмы оказываются более эффективными, чем алгоритмы, разработанные традиционными методами.

Метод декомпозиции получил широкое применение не только при разработке алгоритмов, но и в проектировании электронных схем, построении математических доказательств и в других сферах.

### *1.3.2. Динамическое программирование*

*Динамическое программирование*, как правило, применяется к задачам оптимизации (optimization problems). В таких задачах возможно наличие многих решений. Каждому варианту решения можно сопоставить какое-то значение, и нам нужно найти среди них решение с оптимальным (минимальным или максимальным) значением. Назовем такое решение одним из возможных оптимальных решений. В силу того, что таких решений с оптимальным значением может быть несколько, следует отличать их от единственного оптимального решения. В основе теории динамического программирования лежит принцип оптимальности Беллмана.

Процесс разработки алгоритмов динамического программирования можно разбить на четыре этапа:

1. описание структуры оптимального решения.
2. рекурсивное определение значения, соответствующего оптимальному решению.
3. вычисление значения, соответствующего оптимальному решению, с помощью метода восходящего анализа.
4. составление оптимального решения на основе информации, полученной на предыдущих этапах.

Этапы 1-3 составляют основу метода динамического программирования. Этап 4 может быть опущен, если требуется узнать только значение, соответствующее оптимальному решению. На четвертом этапе иногда используется дополнительная информация, полученная на третьем этапе, что облегчает процесс конструирования оптимального решения.

Суть метода можно объяснить на простом примере чисел Фибоначчи.

Вычислить N чисел в последовательности Фибоначчи: 1, 1,2, 3,5, 8,..., где первые два члена равны единице, а все остальные представляют собой сумму двух предыдущих, N меньше 100.

Самый очевидный способ решения задачи состоит в написании рекурсивной функции. При этом на шестом-седьмом десятке программе перестанет хватать временных ресурсов самой современной вычислительной машины. Это происходит по следующим причинам.

Для вычисления, например, E(40) вначале вычисляется E(39) и E(38). Причем E(38) вычисляется заново, без использования значения, которое было вычислено, когда считалось E(39), т е. значение функции при одном и том же значении аргумента считается мною раз. Если исключить повторный счет, то функция станет заметно эффективней. Для этого приходится завести массив, в котором хранятся значения нашей функции:

Сначала массив заполняется значениями, которые не могут быть значениями функции (чаще веет, это -1, но пригоден и 0). При попытке вычислить какое-то значение, программа смотрит, не вычислялось ли оно ранее, и если да, то берет готовый результат.

Этот подход динамического программирования называется подходом «сверху вниз». Он запоминает решенные задачи, но очередность решения задач все равно контролирует рекурсия.

Можно еще более упростить решение, убрав рекурсию вообще. Для этого необходимо сменить нисходящую логику рассуждения (от того, что надо найти, к тривиальному) на восходящую (соответственно наоборот).

### *1.3.3. «Жадные» алгоритмы*

В *жадном алгоритме* (greedy algorithm) всегда делается выбор, который кажется самым лучшим в данный момент – то есть выполняется локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальной задачи.

Жадные алгоритмы не всегда приводят к оптимальному решению, но во многих задачах они дают нужный результат. Как нередко бывает, "жадная стратегия" подчас обеспечивает лишь сиюминутную выгоду, в то время как в целом результат может оказаться неблагоприятным.

Общего критерия оценки применимости жадного алгоритма для решения конкретной задачи не существует, однако, для задач, решаемых жадными алгоритмами, характерны две особенности: во-первых, к ним применим Принцип жадного выбора, а во-вторых, они обладают свойством Оптимальности для подзадач.

К оптимизационной задаче применим принцип жадного выбора, если последовательность локально оптимальных выборов даёт глобально оптимальное решение. В типичном случае доказательство оптимальности следует такой схеме:

1. Доказывается, что жадный выбор на первом шаге не закрывает пути к оптимальному решению: для всякого решения есть другое, согласованное с жадным выбором и не хуже первого.
2. Показывается, что подзадача, возникающая после жадного выбора на первом шаге, аналогична исходной.
3. Рассуждение завершается по индукции.

Существуют задачи, для которых ни один из известных "жадных" алгоритмов не позволяет получить оптимального решения; но имеются менее "жадные" алгоритмы, которые с большой вероятностью позволяют получать "хорошие" решения. Иногда вполне удовлетворительным можно считать "почти оптимальное" решение.

### *1.3.4. Поиск с возвратом*

Иногда приходится иметь дело с задачей поиска оптимального решения, когда невозможно применить ни один из известных методов, способных помочь отыскать оптимальный вариант решения, и остается прибегнуть к последнему средству – полному перебору. Этот раздел посвящен систематическому описанию метода полного перебора, называемого *поиском с возвратом (backtrack)*, а также метода, называемого альфа-бета отсечением, который зачастую позволяет существенно сократить объем операций поиска.

Термин *backtrack* был введен в 1950 году американским математиком Дерриком Генри Лемером.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это

позволяет значительно уменьшить время нахождения решения. Разновидность задачи методом поиска с возвратом является – *метод ветвей и границ.*

Общий алгоритм для метода ветвей и границ это нахождение оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации (заключается в поиске оптимального объекта в конечном множестве объектов). По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Впервые предложили в 1960 году Ленд и Дойг для решения задач целочисленного программирования.

Общая идея метода может быть описана на примере поиска минимума функции на множестве допустимых значений переменной. Функция и переменная могут быть произвольной природы.

Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной).

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной.

Идея метода следующая: если нижняя граница значений функции на рассматриваемой подобласти дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти, то эта подобласть может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную. Любой узел дерева поиска, нижняя граница которого больше значения этой переменной, может быть исключен из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

### *1.3.5. Алгоритмы локального поиска*

Описанная ниже стратегия нередко приводит к оптимальному решению задачи.

1. Начать с произвольного решения.

2. Для улучшения текущего решения примените к нему какое-либо преобразование из некоторой заданной совокупности преобразований. Это улучшенное решение становится новым "текущим" решением.

3. Повторять указанную процедуру до тех пор, пока ни одно из преобразований в заданной их совокупности не позволит улучшить текущее решение.

Результирующее решение может (необязательно) оказаться оптимальным. В принципе, если "заданная совокупность преобразований" включает все преобразования, которые берут в качестве исходного решения и заменяют его каким-либо другим, процесс "улучшений" не закончится до тех пор, пока мы не получим оптимальное решение. Но в таком случае время выполнения пункта 2 окажется таким же, как и время, требующееся для анализа всех решений, поэтому описываемый подход в целом окажется достаточно бессмысленным.

Этот метод имеет смысл лишь в том случае, когда мы можем ограничить нашу совокупность преобразований небольшим ее подмножеством, что дает возможность выполнить все преобразования за относительно короткое время. Если совокупность

преобразований невелика, естественно рассматривать решения, которые можно преобразовывать одно в другое за один шаг, как "близкие". Такие преобразования называются "локальными", а соответствующий метод называется локальным поиском.

Алгоритмы локального поиска проявляют себя с наилучшей стороны как эвристические алгоритмы для решения задач, точные решения которых требуют экспоненциальных затрат времени. Общепринятый метод поиска состоит в следующем. Начать следует с ряда произвольных решений, применяя к каждому из них локальные преобразования до тех пор, пока не будет получено локально-оптимальное решение, т.е. такое, которое не сможет улучшить ни одно преобразование. На основе большинства (или даже всех) произвольных начальных решений будем получать разные локально-оптимальные решения. Одно из них может быть глобально-оптимальным, т.е. лучше любого другого решения.

На практике можно и не найти глобально-оптимального решения, так как количество локально-оптимальных решений может оказаться колоссальным. Однако можно выбрать локально-оптимальное решение, имеющее минимальную стоимость среди всех найденных решений.

## 2. ОСНОВНЫЕ ЭТАПЫ АЛГОРИТМИЗАЦИИ

*Алгоритмизация* – это процесс разработки и описания алгоритма решения задачи.

В наиболее общем случае алгоритмизация включает следующие основные этапы:

– постановка задачи;

– построение математической модели (моделирование и формализация задачи);

– разработка алгоритма решения задачи;

– программирование.

Ниже излагается сущность основных этапов алгоритмизации задач, решаемых на ЭВМ.

Для небольших программ каждый из таких этапов упрощается, но содержание и последовательность этапов не изменяются.

### 2.1. Постановка задачи

Осуществляется прикладными математиками (алгоритмистами) на основании содержательного описания, включающего краткую содержательную формулировку задачи, официально оформляемых, как правило, в составе технического задания на разработку задачи, решаемой на ЭВМ. Краткая содержательная формулировка представляет собой основу для строгой постановки задачи.

Постановка – это формулировка задачи, определяющая все входные и выходные переменные, а также основные исходные допущения и ограничения. Разработку строгой постановки задачи рекомендуется осуществлять в следующей последовательности:

– выявление состава выходных переменных (определяется характером задачи и потребностями конкретных лиц, заинтересованных в её решении);

– принятие исходных ограничений (вводятся, исходя из специфики и удобств применения задачи, решаемой на ЭВМ.);

– определение состава входных переменных и исходных допущений;

– формулирование строгой постановки задачи (т.е. совокупность значений выходных и входных переменных и перечень основных исходных допущений и ограничений).

К основным характеристикам функциональных задач, уточняемым в процессе ее формализованной постановки, относятся:

– цель или назначение задачи, ее место и связи с другими задачами;

– условия решения задачи с использованием компьютеров;

– содержание функций обработки входной информации при решении задачи;

– требования к периодичности решения задачи;

– ограничения по срокам и точности выходной информации;

– состав и форма представления выходной информации;

– источники входной информации для решения задачи;

– пользователи задачи (кто осуществляет ее решение и пользуется результатами решения).

*Основное требование к постановке задачи – достаточное количество информации для решения задачи*.

*Входная информация* по задаче определяется как данные, поступающие на вход задачи и используемые для ее решения. В качестве входной информации выступают: первичные

данные документов ручного заполнения, файлы базы данных, результаты решения других задач, нормативно-справочная информация, классификаторы, кодификаторы, справочники.

*Выходная информация* по задаче – состав, вид и структура данных, представленных в виде: документов, отчетов, файлов, изображений на экране монитора и пр.

Рассмотрим более подробно сущность процесса определения состава входных переменных и исходных допущений. Данная стадия разработки постановки задачи является наиболее трудной и ответственной. Это обусловлено тем, что свойства реальных объектов зависят от бесконечно большого количества факторов. Учесть все эти факторы, а тем самым определить все входные переменные в строгой постановке задачи и в последующем, при построении математической модели рассматриваемого в задаче объекта или процесса, невозможно. Это приводит к необходимости учёта ограниченного количества существенных факторов, которые отражают самые важные для решаемой задача свойства реальных объектов или процессов. Остальными факторами, следовательно, и свойствами реальных объектов следует пренебречь введением соответствующих допущений.

*Допущения* – это предположения об упрощении реального объекта или процесса, вводимые при формализации задачи.

Специалисты, занимающиеся разработкой строгой постановки задачи, в своем большинстве, полагаются на свой опыт и интуицию, и практика свидетельствует о том, что глубокие и всесторонние знания процессов, подлежащих автоматизации, позволяют правильно определить состав входных переменных и исходных 3 допущений. При этом используется справочная литература, привлекаются результаты, полученные с помощью ранее разработанных математических моделей, проводятся эксперименты, экспертные опросы и т.д.

Когда строгая постановка задачи эвристическим способом (т.е. на основе опыта и интуиции) встречает непреодолимые субъективные затруднения (недостаточны или противоречивы знания о реальных объектах и процессах, не хватает опыта, подводит интуиция и т.д.), можно воспользоваться эмпирическим способом, с использованием методов многомерного статистического анализа (факторный анализ, дисперсионный анализ, корреляционный анализ, регрессионный анализ и т.д.).

Достоинством указанных методов статистического анализа при всех их недостатках, выражающихся, главным образом, в трудоёмкости, является то, что они дают принципиальную возможность научного обоснования постановок задач на основе объективного определения значимости входных переменных.

Постановка задачи заканчивается созданием *технического задания,* а затем и *внешней спецификации программы,* которая включает в себя:

– описание исходных данных и результатов (виды, представление, точность, ограничения и т.п.);

– описание задачи, реализуемой программой;

– способ обращения к программе;

– описание возможных особых и аварийных ситуаций и ошибок пользователя.

На этом этапе программа рассматривается как "черный ящик", для которого определена выполняемая им функция, входные и выходные данные.

## 2.2. Построение математической модели

Построение (разработка) математической модели сводится к установлению математических и логических зависимостей, отражающих физическую сущность рассматриваемого в задаче объекта (процесса) и позволяющих вычислять значения всех

выходных переменных величин, указанных в строгой постановке задачи, при заданных конкретных значениях входных переменных. К разрабатываемым моделям должны быть предъявлены два основных требования – достаточная содержательность и по возможности наибольшая (разумная) простота.

Построение математической модели выполняется в два этапа:

1) декомпозиция моделируемого объекта или процесса на элементы;

2) синтез математических моделей элементов и объекта или процесса в целом.

***Декомпозиция моделируемого объекта*** представляет собой выделение на основе проводимого анализа в рассматриваемом объекте (или процессе) групп однотипных моделируемых элементов, свойствами которых (с учётом принимаемых допущений) обусловлены значения выходных переменных, указанных в строгой постановке задачи.

В наиболее развёрнутом виде декомпозиция моделируемого объекта (процесса) сопровождается оптимизацией и кроме выделения элементов модели включает:

– построение исходных графов состояний моделируемых элементов объекта (процесса);

– оптимизация графов состояний с целью их упрощения и обеспечения (в случае возможности) уменьшения количества состояний, учитываемых при дальнейшей алгоритмизации;

– выявление избыточных графов, которые могут не учитываться при построении

– математической модели;

– выделение опорных графов, которые должны учитываться при построении

– математической модели.

Графы должны отображать учитываемые при построении модели состояния элементов и возможные переходы из одних состояний в другие.

Состояния элементов – это абстрактные аналоги их реальных состояний, различимых при моделировании.

Состояния определяются, исходя из свойств реальных элементов, определяемых перечнем входных и выходных переменных, предусмотренных строгой постановкой задачи, и принятыми допущениями.

Графы состояний наряду с основными допущениями всегда отражают большее количество неоговоренных допущений. Поэтому графы состояний должны согласовываться со специалистами, в интересах которых разрабатывается задача, а основные из них обязательно приводиться в документации.

***Синтез математических моделей*** представляет собой составление математических и логических зависимостей, описывающих рассматриваемые элементы моделируемого объекта (процесса) и их взаимосвязь (взаимодействие) друг с другом.

Результатом синтеза является получение математической модели рассматриваемого в задаче объекта или процесса.

Синтез может осуществляться двумя принципиально различными методами, которые могут применяться и в сочетании. Один из них в дальнейшем называется методом интеграции, а другой – методом дифференциации. Соответственно этим двум методам разработанную на основе полученной математической модели задачу, решаемую на ЭВМ, принято называть методикой машинного расчёта (ММР), либо методикой машинного моделирования (МММ).

Метод дифференциации предполагает отыскание математических и логических зависимостей, описывающих развитие моделируемого процесса (поведение объекта) через смену состояний его элементов в последовательные моменты времени. При данном методе в

качестве промежуточных переменных используются обозначения состояний элементов (применяются кодирование состояний), при этом состояние любого элемента в очередной момент времени описывается функцией его состояния и состояний других элементов в предшествующий момент времени.

В отличие от этого, при использовании метода интеграции в состав промежуточных переменных обозначения состояний элементов не включают. При этом методе синтез математической модели сводится к отысканию аналитических зависимостей, описывающих непосредственную взаимосвязь между входными, промежуточными и выходными переменными, наиболее часто в виде непрерывных функций.

## 2.3. Разработка алгоритма решения задачи

Разработка алгоритма – это определение совокупности предписаний, задающих последовательность действий, которые надо выполнить при варьируемых значениях входных переменных для получения значений выходных переменных в соответствии с найденными при построении математической модели математическими и логическими зависимостями. В результате построения алгоритма осуществляется переход от аналитической модели к имитационной математической модели. Реализация алгоритма приводит к воспроизведению (имитации) свойств моделируемого объекта или процесса.

Построение алгоритма в общем случае включает:

– приведение расчётных соотношений к виду, реализуемому на ЭВМ;

– построение схемы алгоритма и подготовку текстуального описания.

Приведение расчётных соотношений к виду, реализуемому на ЭВМ, осуществляется при необходимости и сводится к их выражению через операции и процедуры (функции), осуществимые в ЭВМ, при этом широко применяются численные методы математики.

При приведении расчётных соотношений к виду, реализуемому на ЭВМ, надо иметь в виду, что большое количество функций для многих типов ЭВМ может быть реализовано с помощью уже созданных стандартных программ. Поэтому перед построением алгоритма, предназначенного для реализации на ЭВМ конкретного типа, полезно ознакомиться с составом имеющейся библиотеки стандартных программ.

Построение схемы алгоритма машинного расчёта на основе аналитической модели, синтезированной методом интеграции, обычно трудностей не вызывает, так как сводится к отображению с помощью символов, установленных ГОСТ, порядка вычислений по формулам расчётных соотношений.

Алгоритм машинного расчёта обычно включает две основные части – задающую и вычислительную. Задающая часть является вспомогательной и содержит начальные предписания алгоритма, предопределяющие подготовку исходных данных и установку начальных значений переменных. Вычислительная часть является основной. Она содержит предписания, определяющие действия по расчёту и выводу значений выходных переменных.

Построение алгоритма заканчивается подготовкой его текстуального описания, которое поясняет используемые математические и логические зависимости и предусматриваемый порядок действий при реализации соответствующей математической модели на ЭВМ.

## 2.4. Программирование

Программирование является формальной записью алгоритма средствами языка программирования и включает в себя:

– выбор языка программирования;
– уточнение способов организации данных;
– запись алгоритма на выбранном языке программирования.

После чего необходимо протестировать написанную программу.

Тестирование выполняется путем вывода промежуточных результатов работы программы и сравнения их с контрольным примером. Для этого либо используют специальные средства отладки программ, имеющиеся в интегрированной среде языка программирования, либо временно добавляют в программу команды вывода промежуточных значений. Уменьшить трудоемкость поиска ошибок в программе можно более тщательным проектированием алгоритма и планированием процесса тестирования на ранних стадиях разработки программы.

Хотя этот этап рассматривается последним, это не значит, что тестирование не должно проводиться на предыдущих этапах. Проектирование и программирование должны обязательно сопровождаться написанием набора тестов – проверочных исходных данных и соответствующих им наборов эталонных реакций.

Необходимо различать процессы тестирования и отладки программы. *Тестирование* представляет собой процесс, посредством которого проверяется правильность функционирования программы и соответствие всем проектным спецификациям. Таким образом, тестирование носит позитивный характер. *Отладка* - процесс исправления ошибок в программе. При отладке, в частности, исправляют ошибки, обнаруженные при тестировании. Следует заметить, что процесс обнаружения ошибок подчиняется экспоненциальному закону, т.е. большинство ошибок обнаруживается на ранних стадиях тестирования и, чем меньше в программе осталось ошибок, тем дольше придется искать каждую из них.

В сложных программах может быть недостаточно тестирования для устранения всех ошибок. Очень часто они обнаруживаются на стадии эксплуатации, поэтому необходим анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов разработки.

## 3. ЭТАПЫ РАБОТ ПО СОЗДАНИЮ ПРОГРАММНЫХ ПРОДУКТОВ

При традиционной неавтоматизированной разработке программ, независимо от принятого метода проектирования и используемого инструментария, выполняются работы, описание которых приведено ниже.

### 3.1. Составление технического задания на программирование

Данная работа соответствует этапу анализа и спецификации программ жизненного цикла программных продуктов.

При составлении технического задания требуется:

– определить платформу разрабатываемой программы - тип операционной системы;

– оценить необходимость сетевого варианта работы программы;

– определить необходимость разработки программы, которую можно переносить на различные платформы;

– обосновать необходимость работы с базами данных под управлением СУБД.

Эти вопросы тесно связаны между собой. На этом же этапе осуществляется:

– выбор методов решения задачи;

– разрабатывается обобщенный алгоритм решения комплекса задач;

– функциональная структура алгоритма или состав объектов;

– определяются требования к комплексу технических средств системы обработки информации;

– интерфейсу конечного пользователя.

### 3.2. Технический проект

На данном этапе выполняется комплекс наиболее важных работ, а именно:

– с учетом принятого подхода к процессу проектирования программного продукта разрабатывается детальный алгоритм реализации функций обработки данных или уточняется состав свойств, методов обработки, событий, запускающих методы обработки;

– определяется состав общесистемного программного обеспечения, включающий базовые средства программного обеспечения (операционная система, СУБД, электронные таблицы и функциональные программные продукты и т.п.),

– разрабатывается внутренняя структура программного продукта, образованная отдельными программными модулями;

– осуществляется выбор инструментальных средств разработки программных модулей.

### 3.3. Рабочая документация (рабочий проект)

На данном этапе осуществляется адаптация базовых средств программного обеспечения (операционной системы, СУБД, инструментальных сред конечного пользователя - текстовых редакторов, электронных таблиц и т.п.). Выполняется разработка программных модулей или методов обработки объектов. Проводится автономная и комплексная отладка программного продукта, испытание работоспособности программных модулей и базовых программных средств. Для комплексной отладки готовится контрольный

пример, который позволяет проверить соответствие возможностей программного продукта заданным спецификациям.

Основным результатом работ этого этапа является также создание основной эксплуатационной документации на программный продукт в составе таких документов, как:

– описание применения - содержит общую характеристику программного изделия с указанием сферы его применения, требований к базовому программному обеспечению, комплексу технических средств;

– руководство пользователя - содержит детальное описание функциональных возможностей и технологии работы с программным продуктом. Данный вид документации ориентирован на конечного пользователя и должен содержать необходимую информацию для самостоятельного освоения и нормальной работы пользователя (с учетом требуемой квалификации пользователя);

– руководство программиста (оператора) - содержит описание особенностей установки (инсталляции) программного продукта и его внутренней структуры - состав и назначение модулей, описывает правила эксплуатации и обеспечения надежной и качественной работы программного продукта.

В ряде случаев на данном этапе для программных продуктов массового применения создаются обучающие системы, демоверсии, гипертекстовые системы помощи.

### 3.4. Ввод в действие

Готовый программный продукт сначала проходит опытную эксплуатацию (пробный рынок продаж), а затем сдается в промышленную эксплуатацию (тиражирование и распространение программного продукта).

# 4. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНЫХ ПРОДУКТОВ

Жизненный цикл программного обеспечения определяется как период времени, который начинается с момента принятия решения о необходимости создания программного обеспечения и заканчивается в момент его полного изъятия из эксплуатации. Основной нормативный документ – международный стандарт ISO/IEC 12207: 1995 «Information Technology – Software Life Cycle Processes».

Программное обеспечение (программный продукт) – набор компьютерных программ, процедур и, возможно, связанной с ними документации и данных.

Процесс – совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные.

Каждый процесс характеризуется:

– определёнными задачами и методами их решения;

– исходными данными, полученными от других процессов;

– результатами.

Основные документы в РФ по разработке программного обеспечения изложены в ГОСТ ЕСПД (19.XXX) – Единая система программной документации, можно выделить следующие:

– ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания»

– ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы»

– ГОСТ 34.603-92 «Информационная технология. Виды испытаний автоматизированных систем».

Программы любого вида характеризуются жизненным циклом, состоящим из отдельных этапов:

1. маркетинг рынка программных средств, спецификация требований к программному продукту;
2. проектирование структуры программного продукта;
3. программирование (создание программного кода), тестирование, автономная и комплексная отладка программ;
4. документирование программного продукта, подготовка эксплуатационной и технологической документации;
5. выход на рынок программных средств, распространение программного продукта;
6. эксплуатация программного продукта пользователями;
7. сопровождение программного продукта;
8. снятие программного продукта с продажи, отказ от сопровождения.

На рисунке 12 изображены этапы жизненного цикла и показаны их временное соответствие друг другу.

Рассмотрим содержание отдельных этапов жизненного цикла.

Рис. 12. Этапы жизненного цикла программного продукта

## 4.1. Маркетинг и спецификация программного продукта

Предназначены для изучения требований к создаваемому программному продукту, а именно:

– определение состава и назначения функций обработки данных программного продукта;

– установление требований пользователя к характеру взаимодействия с программным продуктом, типу пользовательского интерфейса (система меню, использование манипулятора мышь, типы подсказок, виды экранных документов и т.п.);

– требования к комплексу технических и программных средств для эксплуатации программного продукта и т.д.

На данном этапе необходимо выполнить формализованную постановку задачи.

Если программный продукт создается не под заказ и предполагается выход на рынок программных средств, маркетинг выполняется в полном объеме: изучаются программные продукты-конкуренты и аналоги, обобщаются требования пользователей к программному продукту, устанавливается потенциальная емкость рынка сбыта, дается прогноз цены и объема продаж. Кроме того, важно оценить необходимые для разработки программного продукта материальные, трудовые и финансовые ресурсы, ориентировочные длительности основных этапов жизненного цикла программного продукта.

Если программный продукт создается как заказное программное изделие для определенного заказчика, на данном этапе также важно правильно сформулировать и документировать задание на его разработку. Ошибочно понятое требование к программному продукту может привести к нежелательным результатам в процессе его эксплуатации.

Процесс приобретения программного продукта включает следующие действия заказчика:

1) инициирование приобретения:

– определение потребностей в приобретении, разработке или усовершенствовании программного продукта;

– анализ требований к системе;

– принятие решения на приобретение, разработку или усовершенствование существующего программного продукта;

– проверка наличия необходимой документации, гарантий, лицензий, сертификатов и поддержки в случае приобретения программного продукта;

– подготовка плана приобретения (требования к системе, тип договора, ответственность сторон и т.д.);

2) подготовка заявочных предложений:

– требования к системе;

– перечень программных продуктов;

– условия и соглашения;

– технические ограничения (среда функционирования и т.д.);

3) подготовка и корректировка договора:

– определение заказчиком процедуры выбора поставщика, определение критериев оценок предложений возможных поставщиков;

– выбор конкретного поставщика;

– подготовка и заключение договора;

– внесение изменений (при необходимости) в договор в процессе его выполнения;

4) надзор за деятельностью поставщика: осуществляется в соответствии с действиями, предусмотренными в процессах совместной оценки и аудита;

5) приёмка и завершение работ:

– подготавливаются и выполняются необходимые тесты;

– завершение работ осуществляется в случае удовлетворения всех условий приёмки.

Процесс поставки программного продукта охватывает действия поставщика:

1) инициирование поставки: рассмотрение заявочных предложений, принятие решения о согласии с выставленными требованиями и условиями или предложение своих;

2) подготовка ответа на заявочные предложения;

3) подготовка договора;

4) планирование:

– принятие решения на выполнение работы своими силами или с привлечением субподрядчика;

– разработка плана управления проектом, содержащего организационную структуру проекта, разграничение ответственности, технические требования к среде разработки и ресурсам, управление субподрядчиком и др;

5) выполнение и контроль;

6) проверка и оценка;

7) поставка и завершение работы поставщика.

### 4.2. Проектирование структуры программного продукта

Проектирование структуры связано с алгоритмизацией процесса обработки данных, детализацией функций обработки, разработкой структуры программного продукта (архитектуры программных модулей), структуры информационной базы (базы данных) задачи, выбором методов и средств создания программных технологий программирования.

### 4.3. Программирование, тестирование и отладка программ

Это являются технической реализацией проектных решений и выполняются с помощью выбранного инструментария разработчика (алгоритмические языки и системы программирования, инструментальные среды разработчиков и т.п.).

Для больших и сложных программных комплексов, имеющих развитую модульную структуру построения, отдельные работы данного этапа могут выполняться параллельно, обеспечивая сокращение общего времени разработки программного продукта. Важная роль принадлежит используемым при этом инструментальным средствам программирования и отладки программ, поскольку они влияют на трудоемкость выполнения работ, их стоимость, качество создаваемых программ.

Процесс разработки можно разбит на этапы:

1) подготовительная работа:

– выбор модели жизненного цикла программного продукта;

– согласование с заказчиком стандартов, методов и средств разработки;

– составление плана выполнения работ;

2) анализ требований к системе:

– определение её функциональных возможностей;

– определения пользовательских требований;

– определение требований к надёжности и безопасности;

– определение требований к внешним интерфейсам и т.д.;

3) проектирование архитектуры системы: определение компонентов её оборудования, программного обеспечения и операций, выполняемых персоналом;

4) анализ требований к программному обеспечению:

– определение функциональных возможностей, в т.ч. производительности и среды функционирования;

– определение внешних интерфейсов;

– определение спецификаций надёжности и безопасности;

– определение эргономических требований;

– определение требований к используемым данным;

– определение требований к установке и приёмке;

– определение требований к пользовательской документации;

– определение требований к эксплуатации и сопровождению;

5) проектирование архитектуры программного обеспечения:

– трансформация требований к программному обеспечению в архитектуру, определяющую на высоком уровне структуру программного обеспечения и состав его компонентов;

– разработка и документирование программных интерфейсов программного обеспечения и баз данных;

– разработка предварительной версии пользовательской документации;

– разработка и документирование предварительных требований к тестам и плана интеграции программного обеспечения;

6) детальное проектирование программного обеспечения:

– описание компонентов программного обеспечения и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и тестирования;

– разработка и документирование детального проекта базы данных;

– обновление (при необходимости) пользовательской документации;

– разработка и документирование требований к тестам и плана тестирования компонентов программного обеспечения;

– обновление плана интеграции программного обеспечения;

7) кодирование и тестирование программного обеспечения:

– разработка (кодирование) и документирование каждого компонента программного обеспечения и базы данных, а также совокупности тестовых процедур и данных для их тестирования;

– тестирование каждого компонента программного обеспечения и базы данных на соответствие предъявляемым к ним требованиям, документирование результатов тестирования компонентов;

– обновление (при необходимости) пользовательской документации;

– обновление плана интеграции программного обеспечения;

8) интеграция программного обеспечения:

– сборка разработанных компонентов программного обеспечения в соответствии с планом интеграции;

– тестирование агрегированных компонентов;

– разработка тестов и тестовых процедур для проверки каждого из квалификационных требований при последующем квалификационном тестировании;

*Квалификационное требование* – набор критериев или условий, которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

9) квалификационное тестирование программного обеспечения:

– проводится разработчиком в присутствии заказчика для демонстрации того, что программное обеспечение удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации (выполняется для каждого компонента программного обеспечения по всем разделам требований);

– проверяется полнота технической и пользовательской документации и её адекватность самим компонентам программного обеспечения;

10) интеграция системы:

– сборка всех компонентов системы, включая программное обеспечение и оборудование;

– квалификационное тестирование системы в целом на соответствие совокупности требований к ней;

– оформление и проверка полного комплекта документации на систему;

11) установка программного обеспечения:

– осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором;

– проверяется работоспособность программного обеспечения и базы данных;

– если устанавливаемое программное обеспечение заменяет существующую систему, разработчик должен обеспечить их параллельное функционирование;

12) приёмка программного обеспечения (проводится заказчиком с помощью разработчика):

– оценка результатов квалификационного тестирования программного обеспечения и системы;

– документирование результатов оценки;

– разработчик обеспечивает окончательную передачу программного обеспечения заказчику в соответствии с договором, обеспечив необходимое обучение и поддержку.

## 4.4. Документирование программного продукта

Документирование является обязательным видом работ, выполняемых, как правило, не самим разработчиком, а лицом, связанным с распространением и внедрением программного продукта. Документация должна содержать необходимые сведения по установке и обеспечению надежной работы программного продукта, поддерживать пользователей при выполнении функций обработки, определять порядок комплексирования программного продукта с другими программами. Успех распространения и эксплуатации программного продукта в значительной степени зависит от качества его документации.

На машинном уровне программного продукта, как правило, создаются:

– автоматизированная контекстно-зависимая помощь (HELP);

– демонстрационные версии, работающие в активном режиме по типу обучающих систем (электронный учебник) или пассивном режиме (ролик, мультфильм) – для демонстрации функциональных возможностей программного продукта и информационной технологии его использования.

При разработке программных средств создается большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками, как средство управления разработкой и как средство передачи пользователям информации, необходимой для применения и сопровождения программных средств.

На создание этой документации приходится большая доля стоимости программных средств. Документацию можно разбить на две группы:

1. документы управления разработкой программных средств;
2. документы, входящие в состав программных средств.

***Документы управления разработкой*** (*process documentation*), протоколируют процессы разработки и сопровождения, обеспечивая связи внутри коллектива разработчиков и между коллективом разработчиков и менеджерами (лицами, управляющими разработкой).

Эти документы могут быть следующих типов:

1. планы, оценки, расписания – они создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения;

2. отчеты об использовании ресурсов в процессе разработки – создаются менеджерами;

3. стандарты – эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки. Стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка данного программного средства.

4. рабочие документы – это основные технические документы, обеспечивающие связь между разработчиками. Содержат идеи и проблемы, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов.

5. заметки и переписка – эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

***Документы, входящие в состав программных средств*** (*product documentation*), описывают программы как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением программных средств).

Эти документы будут использоваться не только на стадии эксплуатации (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами). Они должны быть проверены

(протестированы) на соответствие программам. Образуют два комплекта с разным назначением:

1. пользовательская документация (П-документация);
2. документация по сопровождению программных средств (С-документация)

***Пользовательская документация*** *(user documentation)* объясняет пользователям, как они должны действовать, чтобы применить данное программное средство. Она необходима, если программное средство предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми руководствуется пользователь при инсталляции (при установке с соответствующей настройкой на среду применения), при применении программного средства для решения своих задач и при управлении. Эти документы частично затрагивают вопросы сопровождения, но не касаются вопросов, связанных с модификацией программ.

***Документация по сопровождению*** (*system documentation*) описывает программное средство с точки зрения ее разработки.

Эта документация необходима, если программное средство предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ. В случае необходимости модернизации программного продукта к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией, которая определяла деятельность команды первоначальных (основных) разработчиков, с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Команда разработчиков-сопроводителей должна будет изучать эту документацию, чтобы понять строение и процесс разработки модернизируемого программного средства, и внести в эту документацию необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное программное средство.

Документация по сопровождению можно разбить на две группы:

1. документация, определяющая строение программ и структур данных программного средства и технологию их разработки;
2. документацию, помогающую вносить изменения в программное средство.

*Первая группа* содержит итоговые документы каждого технологического этапа разработки программного продукта. Она включает следующие документы:

– внешнее описание программного средства (Requirements document);

– описание архитектуры программного средства (description of the system architecture); включая внешнюю спецификацию каждой ее программы. Для каждой программы - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля. Для каждого модуля - его спецификация и описание его строения (design description);

– тексты модулей на выбранном языке программирования (program source code listings);

– документы установления достоверности программного средства (validation documents), описывающие, как устанавливалась достоверность каждой программы и как информация об установлении достоверности связывалась с требованиями к программному средству. Включают, прежде всего, документацию по тестированию (схема тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки ПС, например, доказательства свойств программ.

*Вторая группа* содержит руководство по сопровождению (system maintenance guide), которое описывает известные проблемы вместе с программным средством, описывает, какие

части системы являются аппаратно и программно зависимыми, и как развитие программного средства принято в расчет в его строении (конструкции).

Общая проблема сопровождения это обеспечение, чтобы все представления шли в ногу (оставались согласованными), когда программный продукт изменяется. Чтобы это выполнить, связи и зависимости между документами и их частями должны быть зафиксированы в базе данных управления конфигурацией.

По своему назначению документацию можно классифицировать:

1. технологическая документация процесса разработки, включающая подробные технические описания для специалистов, ведущих проектирование, разработку и сопровождение, обеспечивающую возможность отчуждения, детального освоения, развития и корректировки ими программ и баз данных на всем жизненном цикле программного продукта. Включает в себя:

- проектную документацию, включает:
  - отчет по обследованию предметной области, для которой предназначен разрабатываемого программного продукта, с описанием комплекса задач;
    - описание концепции проектирования;
    - техническое задание на проектирование;
    - план-график работ;
    - спецификации эскизного и технического проекта;
    - документация на разработанные программные модули пакета;
- документацию тестирования компонентов и комплексов программ;
- документацию испытаний;
- документацию сопровождения и управления конфигурацией

2. эксплуатационная (пользовательская) документация программного продукта, создаваемая для конечных пользователей пакета и позволяющая им осваивать и квалифицированно применять его для решения конкретных прикладных задач.

### 4.5. Выход программного продукта на рынок программных средств

Выход связан с организацией продаж массовому пользователю. Этот этап должен по возможности быть коротким, для продвижения программных продуктов применяются стандартные приемы маркетинга: реклама, увеличение числа каналов реализации, создание дилерской и дистрибьютерной сети, ценовая политика - продажа со скидками, сервисное обслуживание и др.

Требуется постоянная программа маркетинговых мероприятий и поддержки программных продуктов. Как правило, для каждого программного продукта существует своя форма кривой продаж, которая отражает спрос.

Вначале продажа программного продукта идет вверх - возрастающий участок кривой. Затем наступает стабилизация продаж программного продукта. Фирма-разработчик стремится к максимальной длительности периода стабильных продаж на высоком уровне. Далее происходит падение объема продаж, что является сигналом к изменению маркетинговой политики фирмы в отношении данного программного продукта, требуется модификация данного продукта, изменение цены или снятие с продажи.

## 4.6. Эксплуатация и сопровождение программного продукта

Эксплуатация программного продукта идет параллельно с его сопровождением, при этом эксплуатация программ может начинаться и в случае отсутствия сопровождения или продолжаться в случае завершения сопровождения еще какое-то время. После снятия программного продукта с продажи определенное время также может выполняться его сопровождение.

В процессе сопровождения в программы вносятся различные изменения:

– исправление ошибок – корректировка программ, выдающих неправильные результаты в условиях, ограниченных техническим заданием и документацией.

– модернизация – расширение функциональных возможностей или улучшение качества решения отдельных задач в соответствии с новым или дополнительным техническим заданием на программный продукт;

– адаптация, регламентированная документацией, к условиям конкретного использования, обусловленным характеристиками внешней среды или конфигурацией аппаратуры.

Процесс эксплуатации включает следующее:

1) подготовительная работа:

– планирование действий и работ, выполняемых в процессе эксплуатации, установка эксплуатационных стандартов;

– определение процедур локализации и разрешения проблем, возникающих в процессе эксплуатации;

2) эксплуатационное тестирование: осуществляется для каждой очередной редакции программного продукта, после чего она передаётся в эксплуатацию;

3) эксплуатация системы: выполняется в предназначенной для этого среде в соответствии с пользовательской документацией;

4) поддержка пользователей: оказание помощи и консультаций при обнаружении ошибок в процессе эксплуатации программного продукта.

В свою очередь сопровождение:

– предусматривает действия и задачи, выполняемые сопровождающей организацией (службой сопровождения);

– активизируется при изменениях (модификациях) программного продукта и соответствующей документации, вызванных возникшими проблемами или потребностями в модернизации либо адаптации программного продукта.

*Сопровождение* (из IEEE-90) – внесение изменений программного обеспечения в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

Этапы сопровождения:

1) подготовительная работа:

– планирование действий и работ, выполняемых в процессе сопровождения:

– определение процедур локализации и разрешения проблем, возникающих в процессе сопровождения;

2) анализ проблем и запросов на модификацию программного продукта (выполняется службой сопровождения):

– анализ сообщений о возникших проблемах или заявок на модификацию

– определяются характеристики модификации:

1) тип (корректирующая, улучшающая, профилактическая, адаптирующая);

2) масштаб (размеры, стоимость и время реализации);

3) критичность (воздействие на производительность, надёжность, безопасность);

– оценка целесообразности проведения модификации и возможных вариантов её проведения;

– утверждение выбранного варианта модификации;

3) модификация программного продукта:

– определение компонентов программного продукта, их версий и документации, подлежащих модификации;

– внесение изменений в соответствии с правилами процесса разработки;

– при необходимости – изменение документации;

4) проверка и приёмка:

– проверка целостности модифицированной системы;

– утверждение внесённых изменений;

5) перенос программного продукта в другую среду:

– используются имеющиеся или разрабатываются новые средства переноса;

– выполняется конвертирование программ в новую среду;

– параллельная эксплуатация в старой и новой среде до полного завершения работ.

### 4.7. Снятие программного продукта с продажи и отказ от сопровождения

Снятие с продажи происходят, как правило, в случае изменения технической политики фирмы-разработчика, неэффективности работы программного продукта, наличия в нем неустранимых ошибок, отсутствия спроса.

Длительность жизненного цикла для различных программных продуктов неодинакова. Для большинства современных программных продуктов длительность жизненного цикла измеряется в годах (2-3 года). Хотя достаточно часто встречаются на компьютерах и давно снятые с производства программные продукты.

## ВОПРОСЫ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ ПО РАЗДЕЛУ «ОСНОВНЫЕ ПОНЯТИЯ ПРОГРАММИРОВАНИЯ»

1. Понятие алгоритма
2. Классификация алгоритмов
3. Язык программирования
4. Классификация языков
5. Способы записи алгоритмов
6. Алгоритм линейной структуры, пример
7. Алгоритм разветвляющейся структуры, пример
8. Алгоритм циклической структуры, пример
9. Принципы проектирования алгоритмов
10. Структурное проектирование
11. Информационное моделирование программных продуктов
12. Объектно-ориентированное проектирование (ООП)
13. Основные понятия ООП
14. Методы разработки алгоритмов. Классификация
15. Методы декомпозиции
16. Пример метода декомпозиции
17. Динамическое программирование
18. Пример метода динамическое программирование
19. «Жадные» алгоритмы
20. Поиск с возвратом
21. Метод ветвей и границ
22. Пример метода ветвей и границ
23. Ограничение эвристических алгоритмов
24. Алгоритмы локального поиска
25. Основные этапы алгоритмизации
26. Основные этапы алгоритмизации. Постановка задачи
27. Основные этапы алгоритмизации. Построение математической модели
28. Основные этапы алгоритмизации. Разработка алгоритма решения зада
29. Основные этапы алгоритмизации. Программирование
30. Этапы работ по созданию программных продуктов
31. Составление технического задания на программирование
32. Технический проект по созданию программных продуктов
33. Рабочая документация (рабочий проект). Основные виды
34. Жизненный цикл программных продуктов
35. Маркетинг и спецификация программного продукта
36. Проектирование структуры программного продукта
37. Программирование, тестирование и отладка программ
38. Документирование программного продукта
39. Выход программного продукта на рынок программных средств
40. Эксплуатация и сопровождение программного продукта

41. Снятие программного продукта с продажи и отказ от сопровождения

42. Основные виды, этапы проектирования и жизненный цикл программных продуктов

43. Стандарты на разработку. Стандарты на разработку прикладных программных средств. Документирование, сопровождение и эксплуатация программных средств

## РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА


1. http://www.tadviser.ru/index.php/Статья:Решение_задач_на_ЭВМ
2. http://www.cyberguru.ru/programming/cpp/cpp-data-abstraction.html Бьерн Страуструп - Абстракция данных в языке С++
3. http://progstone.narod.ru/links/wantalgo.html
4. http://algolist.manual.ru/
5. IEEE Std 610.12 – 1990/ IEEE Standard of Glossary of Software Engineering Terminology
6. Алексеев Е.Г., Богатырев С.Д. Информатика. Мультимедийный электронный учебник точка доступа http://inf.e-alekseev.ru/index.html
7. Афанасьева Т.В. Основы визуальной алгоритмизации: учеб. пособие. – Ульяновск, УлГТУ. 2002 – 70 с.
8. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979 – 304 с.
9. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы. : Пер. с англ. : учеб. пособие. – М. : Издательский дом "Вильяме", 2000. – 384 с.– ISBN 5-8459-0122-7
10. Вирт Н. Алгоритмы и структуры данных. М: Мир, 1989. – 360 с.
11. Вирт И. Алгоритмы + структуры данных = программы. М.: Мир, 1985. – 204 с.
12. Глушаков С.В., Коваль А.В, Смирнов С.В. Язык программирования С++. Учебный курс. – Харьков: «Фолио», 2001. – 500с.
13. Голицына О.Л., Попов И.И. Основы алгоритмизации и программирования : учеб. пособие. - М: ФОРУМ, 2008 – 432 с.
14. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения
15. Грин Д., Кнут Д. Математические методы анализа алгоритмов. М: Мир, 1987. – 218 с.
16. Давыдов В.Г., Лекарев М.Ф. Учебный машинно-ориентированный язык (ПМ-ассемблер): Учебное пособие. – Санкт-Петербургский государственный технический университет, СПб, 2000. – 36 с.
17. Давыдов, В.Г. Программирование и основы алгоритмизации: учеб. пособие – М.: Высш. шк., 2003. – 447 с.
18. Дональд Кнут. Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol.1. Fundamental Algorithms. – 3-е изд. – М.: «Вильямс», 2006. — С. 720. — ISBN 0-201-89683-4.
19. Зикратов И.А., Косовцев В.В., Петров В.Ю. Разработка и стандартизация программных средств и информационных технологий: учеб. пособие. - СПб.: СПбГУ ИТМО, 2010. - 91 с.
20. Каймин В.А. Информатика. Учебник для студентов.– М., ИНФРА-М, 2009.
21. Кнут Д.Е. Искусство программирования для ЭВМ: В 3 т. М.: Мир, 1976
22. Кнут Д.Е. Искусство программирования: В 3 т. М.: Вильяме, 2000
23. Колдаев В.Д. Основы алгоритмизации и программирования. Учебник по программированию. – М., ИНФРА-М, 2006

24. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И.В. Красикова. – 2-е изд. – М.: Вильямс, 2005. – 1296 с. — ISBN 5-8459-0857-4.

25. Кузниченко М.А. Динамические структуры данных: учебное пособие. – Орск: Издательство ОГТИ, 2011. – 102 с. – ISBN 978-5-8424-0551-0

26. Лэгсам Й., Огенстайн М. Структуры данных для персональных ЭВМ. М.: Мир, 1989. – 586 с.

27. Либерти Джесс. Освой самостоятельно С++ за 21 день.- М.: Издательский Дом “Вильямс”, 2001. – 814 с.

28. Макаров В.Л. Программирование и основы алгоритмизации: учеб. пособие – СПб.: СЗТУ, 2003. – 110 с.

29. Макаров В.Л. Программирование и основы алгоритмизации: учеб. пособие – СПб.: СЗТУ, 2003. – 110 с.

30. Макарова Н.В., Титова Ю.Ф., Гурьева Т.Н., Соколов Н.Е. Электронное учебное пособие по дисциплине «Информатика» предназначено для студентов 1-го курса http://eos.ibi.spb.ru/ pluginfile.php/51723/ mod_page/content/ 1/ 11_15/ index.html АНО ВПО "МБИ" СПб

31. Мозговой М.В. Занимательное программирование: Самоучитель. – СПб.: Питер, 2004. – 150 с.

32. Одинцов И.О. Профессиональное программирование. Системный подход. – 2-е изд. перераб. и доп. – СПб.: БХВ-Петербург, 2004 - 260с.

33. Павловская Т.А. С/С++. Программирование на языке высокого уровня. Учебник. – М.: «Питер», 2002. - 460с.

34. Павловская Т.А. С/С++. Программирование на языке высокого уровня. Учебник. – М.: «Питер», 2002. -460с.

35. Павловская Т.А., Щупак Ю.А. С\С++. Объектно-ориентированное программирование . – СПб.: «Питер», 2005. -264с.

36. Пол Ирэ. Объектно-ориентированное программирование с использованием С++: Пер. с англ. / Ирэ Пол. - К.: НИПФ "ДиаСофт Лтд", 1995. – 480 с.

37. Программирование и основы алгоритмизации: методические указания / сост.: И.А. Елизаров, С.Б. Путин, С.А. Скворцов, А.А. Третьяков, С.И. Татаренко. – Тамбов: Изд-во Тамбовский государственный технический университет, 2007. – 40 с.

38. Программирование на С++: учеб. пособие под ред. А.Д. Хомоненко. – СПБ.: Коронапринт, 1999. – 256 с.

39. Рассохин Д. От Си к Си++. М.: Издательство "ЭДЕЛЬ", 1993. – 128 с.

40. Семакин И.Г. Шестаков А.П. Основы программирования: Учебник для СПО. – М.: Издательский центр «Академия», 2004. – 432 с.

41. Сибуя М., Ямамото Т. Алгоритмы обработки данных. М.: Мир, 1986. – 218 с.

42. Строганов Р.П. Алгоритмическое обеспечение задач управления в АСУ ТП: Учеб. пособие – Л.: ЛПИ , 1989. – 221 с.

43. Тихомиров Ю.В. Visual С++ 6 – СПб. – БХВ – Санкт-Петербург, 1999. – 496 с.

44. Трой Д. Программирование на языке Си для персонального компьютера IBM PC: Пер. с англ. - М.: Радио и связь, 1991.– 406 с.

45. Успежкий В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. М.: Наука, 1987. – 128 с.

46. Франко П. С++: учеб. курс. – СПБ.: Питер, 2002. – 528 с.

47. Шилдт Г. Самоучитель C++. 3-е издание: Пер. с англ. – СПБ.: BHV – Санкт-Петербург, 1998. – 688 с.

48. Шумова Е.О. Программирование на языке высокого уровня: Учеб. пособие. Ч.1. – СПб.: СЗТУ,2001. – 82 с.

49. Юркин А.Г. Задачник по программированию. – СПб.: «Питер», 2002. – 182с.

Учебное текстовое электронное издание

**Самарина Ирина Геннадьевна**
**Бондарева Альбина Робертовна**

**ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ**
**Часть 3**

Учебное пособие

0,57 Мб
1 электрон. опт. диск

г. Магнитогорск, 2015 год
ФГБОУ ВПО «МГТУ»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВПО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра автоматизированных систем управления
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru