

Лекция 4. Разработка структурной и функциональной схем

Сущность структурного подхода заключается в декомпозиции программы или программной системы по функциональному принципу. Все методы декомпозиции рассчитаны на анализ и проектирование как структур данных, так и обрабатывающих их программ. В большинстве случаев первичным считают проектирование обрабатывающих компонентов, проектирование же структур данных выполняют параллельно. Существует и альтернативный подход, при котором первичным считают проектирование данных, а обрабатывающие программы получают, анализируя полученные структуры данных. В любом случае проектирование ПО начинают с определения его структуры.

Структурное и «неструктурное» программирование. Средства описания структурных алгоритмов

Одним из способов обеспечения высокого уровня технологичности разрабатываемого ПО является *структурное программирование*.

Различают три вида вычислительного процесса, реализуемого программами: линейный, разветвленный и циклический.

Линейная структура процесса вычислений предполагает, что для получения результата необходимо выполнить некоторые операции в определенной последовательности.

Разветвленная структура процесса вычислений предполагает, что конкретная последовательность операций зависит от значений одной или нескольких переменных.

Циклическая структура процесса вычислений предполагает, что для получения результата некоторые действия необходимо выполнить несколько раз.

Для реализации указанных вычислительных процессов в программах используют соответствующие управляющие операторы. Первые процедурные языки программирования высокого уровня (такие, как FORTRAN) понятием «тип вычислительного процесса» не оперировали. Для изменения линейной последовательности операторов в них, как в языках низкого уровня, использовались команды условной (при выполнении некоторого условия) и безусловной передач управления. Потому и программы, написанные на этих языках, имели запутанную структуру, присущую в настоящее время только низкоуровневым языкам.

Для изображения схем алгоритмов таких программ был разработан [ГОСТ 19.701-90](#), согласно которому каждой группе действий ставится в соответствие специальный блок (табл. 4.1). Хотя этот стандарт предусматривает блоки для обозначения циклов, он не запрещает и произвольной передачи управления, т.е. допускает использование команд условной и безусловной передачи управления при реализации алгоритма.

Табл. 4.1. Основные блоки схемы алгоритма

Название блока	Обозначение	Назначение блока
Терминатор		Начало, завершение программы или подпрограммы
Процесс		Обработка данных (вычисления, пересылки и т. п.)
Данные		Операции ввода-вывода
Решение		Ветвления, выбор, итерационные и поисковые циклы
Подготовка		Счетные циклы
Граница цикла		Любые циклы
		
Предопределенный процесс		Вызов процедур
Соединитель		Маркировка разрывов линий
Комментарий	--- 	Пояснения к операциям

После того, как в 60-х годах XX в. было доказано, что любой сколь угодно сложный алгоритм можно представить с использованием трех основных управляющих конструкций, в языках программирования высокого уровня появились управляющие операторы для реализации соответствующих конструкций. Эти три конструкции принято считать базовыми. К ним относят конструкции:

- *следование* - обозначает последовательное выполнение действий;
- *ветвление* - соответствует выбору одного из двух вариантов действий;
- *цикл-пока* - определяет повторение действий, пока не будет нарушено некоторое условие, выполнение которого проверяется в начале цикла.

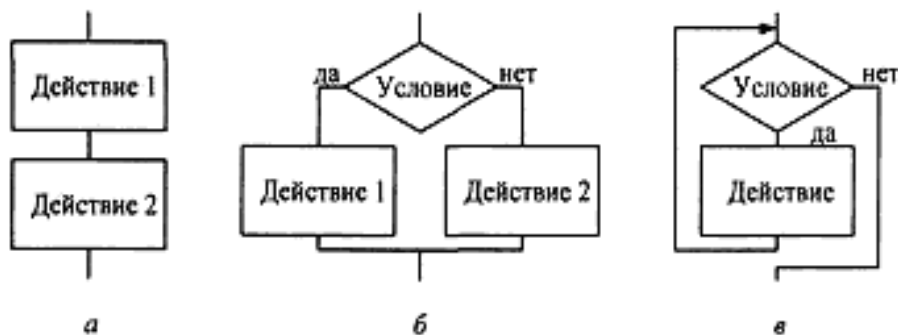


Рис. 4.1. Базовые алгоритмические структуры: а – следование; б – ветвление; в – цикл-пока

Помимо базовых, процедурные языки программирования высокого уровня используют еще три конструкции, которые можно составить из базовых (рис.4.2):

- *выбор* - обозначает выбор одного варианта из нескольких в зависимости от значения некоторой величины;
- *цикл-до* - обозначает повторение некоторых действий до выполнения заданного условия, проверка которого осуществляется после выполнения действий в цикле;
- *цикл с заданным числом повторений* (счетный цикл) - обозначает повторение некоторых действий указанное количество раз.

Любая из дополнительных конструкций легко реализуется через базовые. Все шесть конструкций положены в основу структурного программирования.

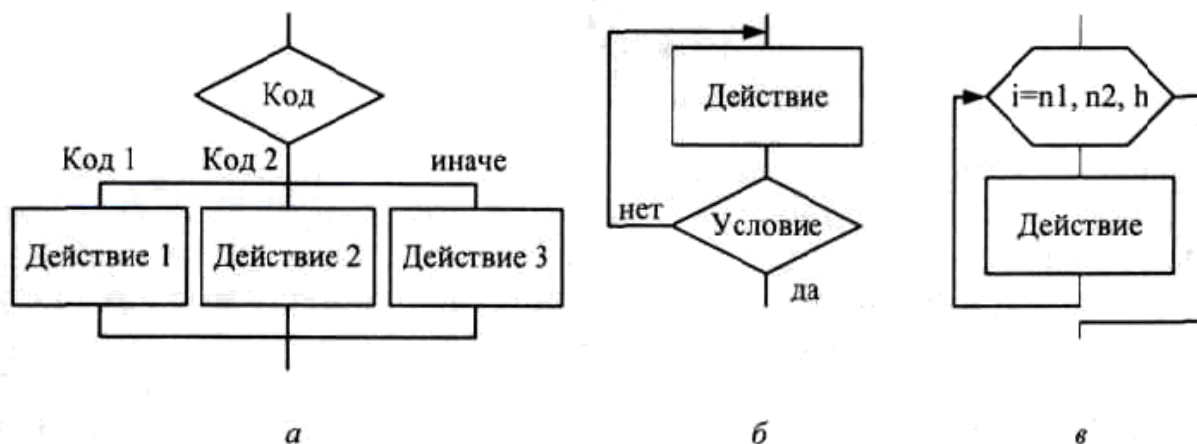


Рис. 4.2. Дополнительные структуры: а – выбор; б – цикл-до;
в – цикл с заданным числом повторений

Слово «структурное» в данном названии подчеркивает факт, что при программировании использованы только перечисленные конструкции (структуры). Отсюда и понятие «программирование без «go to». Программы, написанные с использованием только структурных операторов передачи управления, называют структурными, чтобы подчеркнуть их отличие от программ, при проектировании или реализации которых использовались низкоуровневые способы передачи управления.

Псевдокоды. *Псевдокод* – формализованное текстовое описание алгоритма (текстовая нотация). Существует несколько вариантов псевдокодов. Один из них - в табл. 4.2

Описать с помощью псевдокодов неструктурный алгоритм невозможно. Использование псевдокодов изначально ориентирует проектировщика только на структурные способы передачи управления, а потому требует более тщательного анализа разрабатываемого алгоритма. В отличие от схем алгоритмов, псевдокоды не ограничивают степень детализации проектируемых операций. Они позволяют соизмерять степень детализации действия с уровнем абстракции и согласуются с основным методом структурного программирования – методом пошаговой детализации.

Табл. 4.2. Псевдокод

Структура	Псевдокод	Структура	Псевдокод
Следование	<действие 1> <действие 2>	Выбор	Выбор <код> <код 1>: <действие 1> <код 2>: <действие 2> Все-выбор
Ветвление	Если <условие> то <действие 1> иначе <действие 2> Все-если	Цикл с заданным количеством повторений	Для <индекс> = <n>, <k>, <h> <действие> Все-цикл
Цикл-пока	Цикл-пока <условие> <действие> Все-цикл	Цикл-до	Выполнять <действие> До <условие>

Разработка структурной и функциональной схем

Процесс проектирования сложного ПО начинают с уточнения его структуры - определения структурных компонентов и связей между ними. Результат уточнения структуры может быть представлен в виде структурной и/или функциональной схем и описания (спецификаций) компонентов.

Структурной называют схему, отражающую *состав и взаимодействие по управлению* частей разрабатываемого ПО.

Структурные схемы пакетов программ не информативны, поскольку организация программ и пакеты не предусматривает передачи управления между ними. Поэтому структурные схемы разрабатывают для каждой программы пакета, а список программ пакета определяют, анализируя функции, указанные в техническом задании.

Самый простой вид ПО - программа, которая в качестве структурных компонентов может включать только подпрограммы и библиотеки ресурсов. Разработку структурной схемы программы обычно выполняют методом пошаговой детализации.

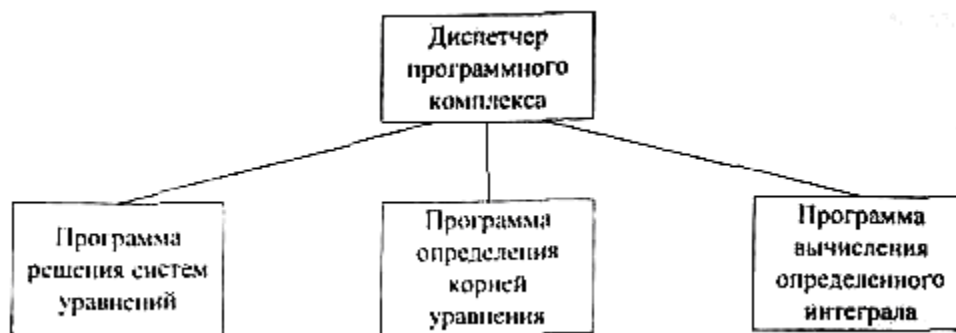


Рис. 4.3. Пример структурной схемы программного комплекса

Структурными компонентами программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиотеки ресурсов и т. п.

Структурная схема программного комплекса демонстрирует передачу управления от программы-диспетчера соответствующей программе (рис. 4. 3).

Структурная схема программной системы показывает наличие подсистем или других структурных компонентов. В отличие от программного комплекса отдельные части (подсистемы) программной системы интенсивно обмениваются данными между собой и, возможно, с основной программой. Структурная схема программной системы этого обычно не показывает (рис. 4.4).

Более полное представление о проектируемом ПО с точки зрения взаимодействия его компонентов между собой и с внешней средой дает функциональная схема.

Функциональная схема или *схема данных* (ГОСТ 19.701 -90) - схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.

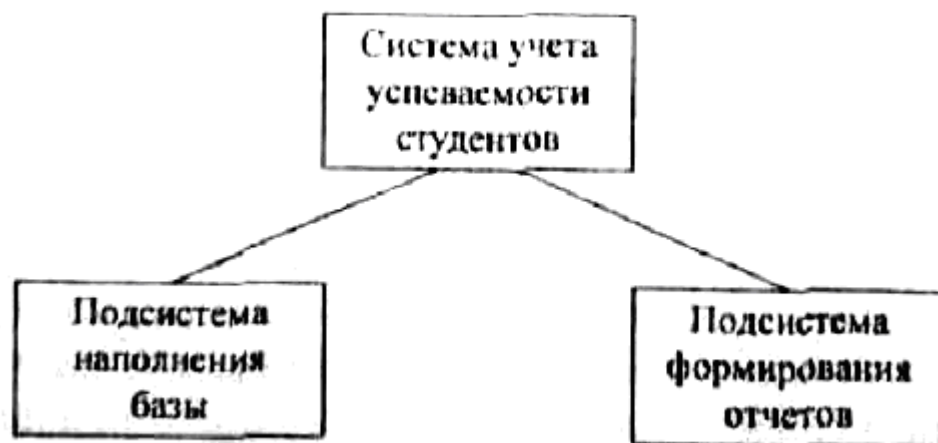










Рис. 4.4. Пример структурной схемы программной системы

Для изображения функциональных схем используют специальные обозначения, установленные стандартом **ГОСТ 19.701-90** (табл. 4.3).

Табл. 4.3.

Название блока	Обозначение	Назначение блока
Запоминаемые данные		Для обозначения таблиц и других структур данных, которые должны быть сохранены без уточнения типа устройства
Оперативное запоминающее устройство		Для обозначения таблиц и других структур данных, хранящихся в оперативной памяти
Запоминающее устройство с последовательной выборкой		Для обозначения таблиц и других структур данных, хранящихся на устройствах с последовательной выборкой (магнитной ленте и т.п.)
Запоминающее устройство с прямым доступом		Для обозначения таблиц и других структур данных, хранящихся на устройствах с прямым доступом (дисках)
Документ		Для обозначения таблиц и других структур данных, выводимых на печатающее устройство
Ручной ввод		Для обозначения ручного ввода данных с клавиатуры
Карта		Для обозначения данных на магнитных или перфорированных картах
Дисплей		Для обозначения данных, выводимых на дисплей компьютера

Функциональные схемы более информативны, чем структурные. На рис. 4.5 для сравнения приведены функциональные схемы программных комплексов и систем.

Все компоненты структурных и функциональных схем должны быть описаны. При структурном подходе тщательно необходимо прорабатывать спецификации межпрограммных интерфейсов, так как от качества их описания зависит количество самых дорогостоящих ошибок.

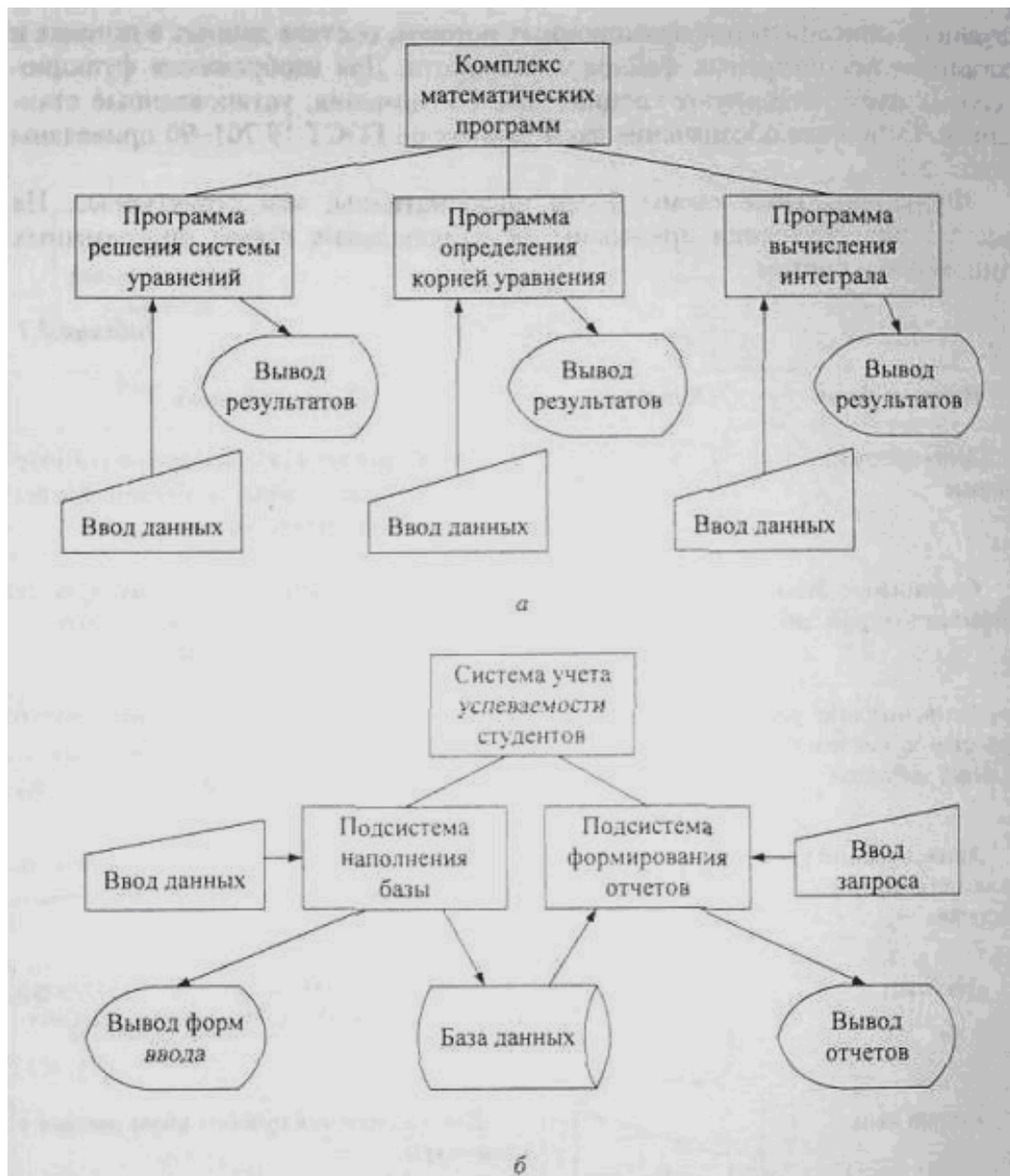


Рис. 4.5. Примеры функциональных схем: а – комплекс программ; б - программная система

Использование метода пошаговой детализации для проектирования структуры ПО

Структурный подход к программированию в том виде, в котором он был сформулирован в 70-х годах XX в., предлагал осуществлять декомпозицию программ методом пошаговой детализации. Результатом декомпозиции является структурная схема программы, которая представляет собой многоуровневую иерархическую схему взаимодействия подпрограмм по управлению.

Минимально такая схема отображает два уровня иерархии, т. е. показывает общую структуру программы. Однако тот же метод позволяет получить структурные схемы с большим количеством уровней.

Метод пошаговой детализации реализует нисходящий подход и базируется на основных конструкциях структурного программирования. Он предполагает пошаговую разработку алгоритма. Каждый шаг при этом включает разложение функции на подфункции. Так на первом этапе описывают решение поставленной задачи, выделяя общие подзадачи, на следующем аналогично описывают решение подзадач, формулируя при этом подзадачи следующего уровня. Таким образом, на каждом шаге происходит уточнение функций проектируемого программного обеспечения. Процесс продолжают, пока не доходят до подзадач, алгоритмы решения которых очевидны.

Декомпозируя программу методом пошаговой детализации, следует придерживаться основного правила структурной декомпозиции, следующего из принципа вертикального управления: в первую очередь детализировать управляющие процессы декомпозируемого компонента, оставляя уточнение операций с данными напоследок. Это связано с тем, что приоритетная детализация управляющих процессов существенно упрощает структуру компонентов всех уровней иерархии и позволяет не отделять процесс принятия решения от его выполнения: так, определив условие выбора некоторой альтернативы, сразу же вызывают модуль, ее реализующий.

Детализация операций со структурами в последнюю очередь позволит отложить уточнение их спецификаций и обеспечит возможность безболезненной модификации этих структур за счет сокращения количества модулей, зависящих от этих данных.

Кроме этого, целесообразно придерживаться следующих рекомендаций:

- не отделять операции инициализации и завершения от соответствующей обработки, так как модули инициализации и завершения имеют плохую связность (временную) и сильное сцепление (по управлению);
- не проектировать слишком специализированных или слишком универсальных модулей, так как проектирование излишне специальных модулей увеличивает их количество, а проектирование излишне универсальных модулей повышает их сложность;
- избегать дублирования действий в различных модулях, так как при их изменении исправления придется вносить во все фрагменты программы, где они выполняются - в этом случае целесообразно просто реализовать эти действия в отдельном модуле;
- группировать сообщения об ошибках в один модуль по типу библиотеки ресурсов, тогда будет легче согласовать формулировки, избежать дублирования сообщений, а также перевести сообщения на другой язык.

При этом, описывая решение каждой задачи, желательно использовать не более 1-2-х структурных управляющих конструкций, таких, как цикл-пока или ветвление, что позволяет четче представить себе структуру организуемого вычислительного процесса.

Пример.1. Разработать алгоритм программы построения графиков функций одной переменной на заданном интервале изменения аргумента $[x_1, x_2]$ при условии непрерывности функции на всем интервале определения.

Примечание. Для того чтобы программировать построение графиков функций с точками разрыва первого и второго рода, необходимо аналитически исследовать заданные функции, что представляет собой отдельную и достаточно сложную задачу. Численными методами данная задача не решается.

В общем виде задача построения графика функции ставится как задача отображения реального графика (рис. 4.6, а), выполненного в некотором масштабе, в соответствующее изображение в окне на экране (рис. 4.6, б).

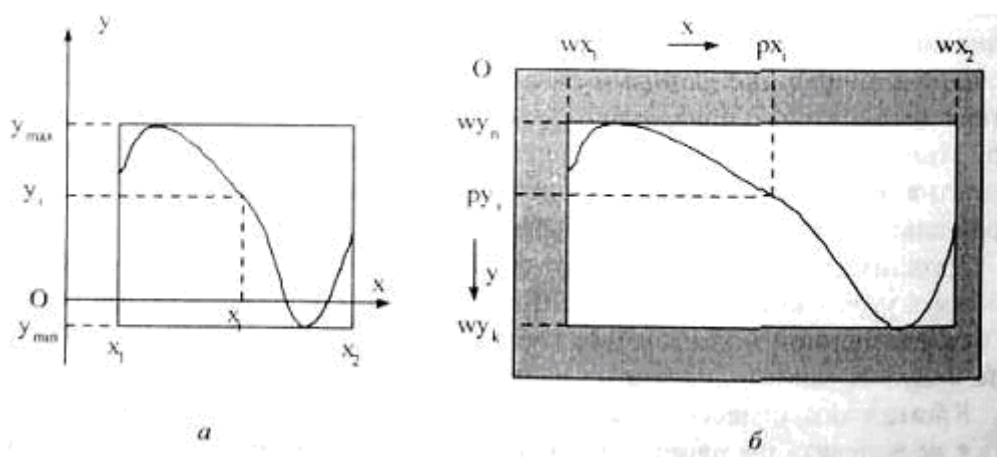


Рис. 4.6. Задача построения графика функции: а – график функции; б – его отображение на экране

Для построения графика необходимо определить масштабы по осям координат:

$$m_x = \frac{wx_1 - wx_2}{x_1 - x_2}, \quad m_y = \frac{wy_1 - wy_2}{y_{\max} - y_{\min}}$$

и координаты точек графика:

$$px_i = \lceil (x_i - x_1) * m_x \rceil + wx_1,$$

$$py_i = \lfloor (y_{\max} - y_i) * m_y \rfloor + wy_1,$$

Шаг координатной сетки по вертикали и горизонтали при этом можно определить по формулам:

$$lpm_x = \frac{wx_2 - wx_1}{nl_x}, \quad lpm_y = \frac{wy_2 - wy_1}{nl_y}$$

где nl_x nl_y - соответственно количество вертикальных и горизонтальных линий.

Для разметки сетки определим шаги разметки по горизонтали и вертикали:

$$lm_x = \frac{x_2 - x_1}{nl_x}, \quad lm_y = \frac{x_{\max} - x_{\min}}{nl_y}$$

Чтобы построить график, необходимо задать функцию, интервал изменения аргумента $[x_1, x_2]$, на котором функция непрерывна, количество точек графика n , размер и положение окна экрана, в котором необходимо построить график: w_{x1} , w_{y1} , w_{x2} , w_{y2} и количество линий сетки по горизонтали и вертикали nl_x , nl_y . Значения w_{x1} , w_{y1} , w_{x2} , w_{y2} , nl_x , nl_y можно задать, исходя из размера экрана, а интервал и число точек графика надо вводить.

Разработку алгоритма выполняем методом пошаговой детализации, используя для записи псевдокод. Примем, что программа будет взаимодействовать с пользователем через традиционное иерархическое меню, которое содержит пункты: *Функция*, *Отрезок*, *Шаг*, *Вид результата*, *Выполнить* и *Выход*. Для каждого пункта этого меню необходимо реализовать сценарий, предусмотренный в техническом задании.

Шаг 1. Определяем структуру управляющей программы, которая реализует работу с меню через клавиатуру:

Программа.

Инициализировать глобальные значения

Вывести заголовок и меню

Выполнять

Если выбрана Команда

то Выполнить Команду

иначе Обработать нажатие клавиши управления

Все - если

до Команда = Выход Конец.

Очистка экрана, вывод заголовка и меню, а также выбор Команды - операции сравнительно простые, следовательно, их можно не детализировать.

Шаг 2. Детализируем операцию Выполнить команду:

Выполнить Команду:

Выбор Команда

Функция:

Ввести или выбрать формулу Fun

Выполнить разбор формулы

Отрезок:

Ввести значения $x1, x2$

Шаг:

Ввести значения h

Вид результата:

Ввести вид _результата

Выполнить:

Рассчитать значения функции

Если Вид_результата = График

то Построить график

иначе Вывести таблицу

Все – если

Все – выбор

Определим, какие фрагменты следует реализовать в виде подпрограмм. Во-первых, фрагмент Вывода заголовка и меню, так как это достаточно длинная линейная последовательность операторов и её выделение в отдельную процедуру позволит сократить управляющую программу. Во-вторых, фрагменты Разбор формулы, Расчет значений функции, Построение графика и Вывод таблицы, так как это достаточно сложные операции. Это – программы первого уровня, которые определяют структуру программы (рис. 4.7).

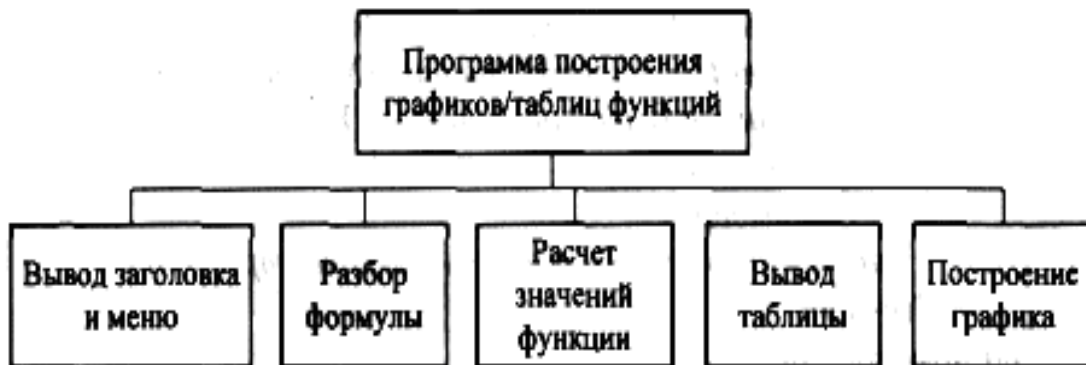


Рис. 4.6. Структурная схема программы построения графика функций

Определим для них интерфейсы по данным с основной программой, т.е. списки параметров. Подпрограмма Вывод заголовка и меню параметров не предполагает.

Подпрограмма Разбор формулы должна иметь два параметра: Fun - аналитическое задание функции, Tree - возвращаемый параметр - адрес дерева разбора.

Подпрограмма Расчет значений функции должна получать адрес дерева разбора Tree, отрезок: значения x_1 и x_2 , а также шаг h . Обратно в программу она должна возвращать таблицу значений функции $X(n)$ и $Y(n)$, где n - количество точек функции.

Подпрограммы Вывода таблицы и Построения графика должны получать таблицу значений функции и количество точек.

После уточнения имен переменных алгоритм основной программы:

Программа.

Вывод заголовка и меню

Выполнять

Если выбрана Команда то

Выбор Команда Функция:

Ввести или выбрать формулу Fun

Разбор формулы (Fun; Var Tree) Отрезок:

Ввести значения x_1, x_2 Шаг:

Ввести значения h

Вид результата:

Ввести вид _результата Выполнить:

Расчет значений функции ($x_1, x_2, k, Tree; VarX, Y, n$)

Если Вид _результата=График то

Построение графика(X, Y, n)

иначе Вывод таблицы(X, Y, n)

Все-если

Все-выбор

иначе Обработать нажатие клавиш управления

Все-если

до Команда=Выход

Конец.

На следующих шагах необходимо выполнить детализацию алгоритмов подпрограмм. Детализацию выполняют, пока алгоритм программы не станет полностью понятен. Один из возможных вариантов полной структурной схемы данной программы показан на рис.4.7.

Использование метода пошаговой детализации обеспечивает высокий уровень технологичности разрабатываемого ПО, так как он позволяет использовать только структурные способы передачи управления.

Разбиение на модули при данном виде проектирования выполняется эвристически, исходя из рекомендуемых размеров модулей (20-60 строк) и сложности структуры (две-три вложенных управляющих конструкции). В качестве модуля (подпрограммы) можно реализовать решение подзадач, сформулированных на любом шаге процесса детализации, однако определяющую роль при разбиении программы на модули играют принципы обеспечения технологичности модулей.

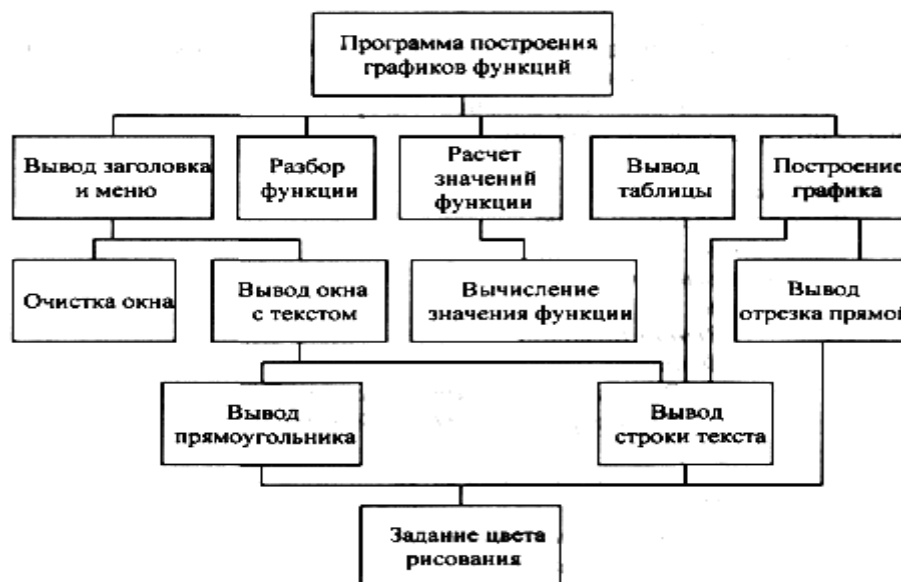


Рис. 4.6. Полная структурная схема программы построения графика функций

Примеры построения схемы алгоритма Шага 1 и Шага 2 приведены на рис. 4.7 и рис.4.8.

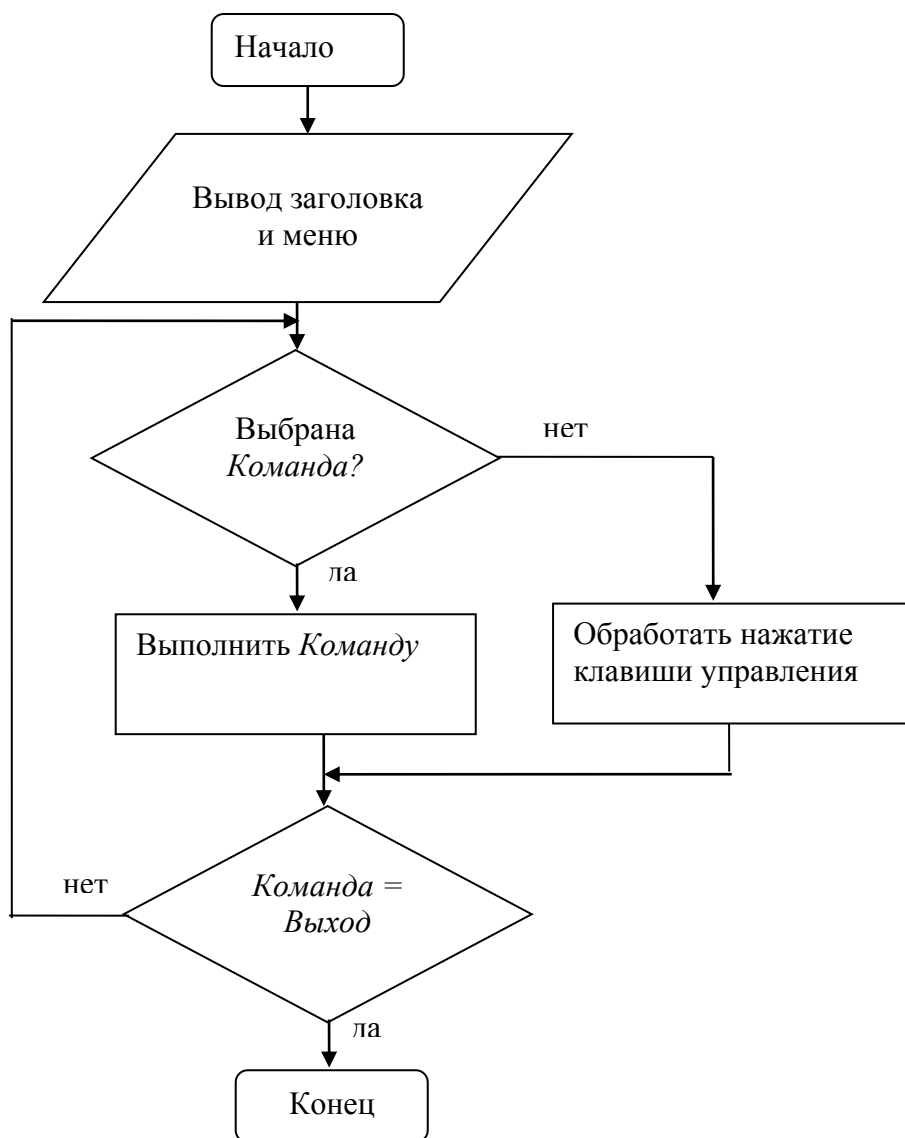


Рис. 4.7. Схема алгоритма Шага 1.

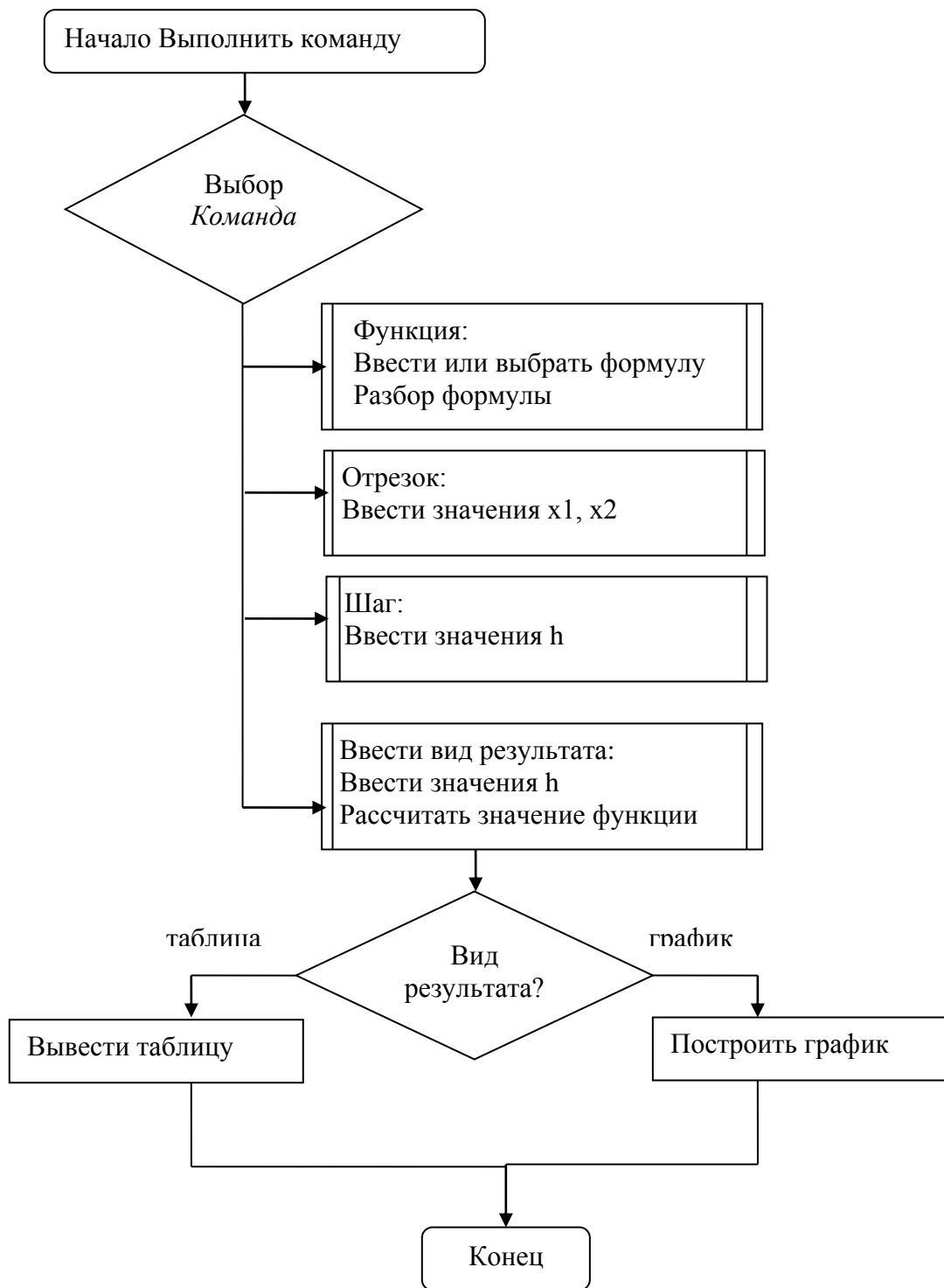


Рис. 4.8. Схема алгоритма Шага 2.