5. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ СТРУКТУРНОМ ПОДХОДЕ

Как уже упоминалось ранее, сущность структурного подхода заключается в декомпозиции программы или программной системы по функциональному принципу. Все предлагаемые методы декомпозиции используют интерфейсы простейшего типа: примитивные интерфейсы и традиционные меню, и рассчитаны на анализ и проектирование как структур данных, так и обрабатывающих их программ.

Причем 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 программного обеспечения начинают с определения его структуры.

5.1. Разработка структурной и функциональной схем

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

Структурная схема разрабатываемого программного обеспечения. *Структурной* называют схему, отражающую *состав и взаимодействие по управлению* частей разрабатываемого программного обеспечения.

Структурные схемы п а к е т о в п р о г р а м м не информативны, поскольку организация программ в 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 программного обеспечения - программа, которая в качестве структурных компонентов может включать только подпрограммы и библиотеки ресурсов. Разработку структурной схемы программы обычно выполняют методом пошаговой детализации (см. § S.2).

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

Структурная схема п р о г р а м м н о г о к о м п л е к с а демонстрирует передачу управления от программы-диспетчера соответствующей программе (рис. 5.1).

Рис. 5.1. Пример структурной схемы программного комплекса

Структурная схема программной системы, как правило, 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 же схема программной системы этого обычно не показывает (рис. 5.2).

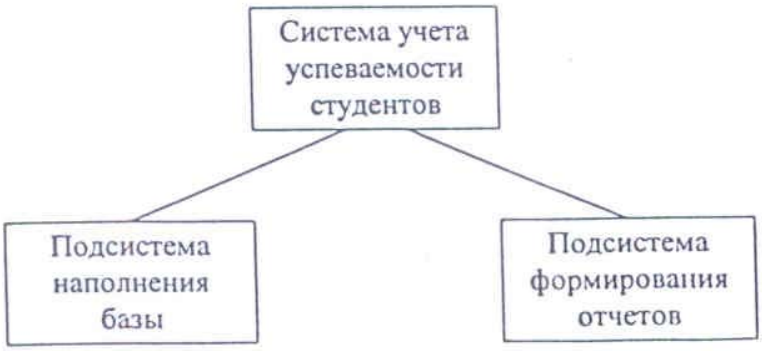


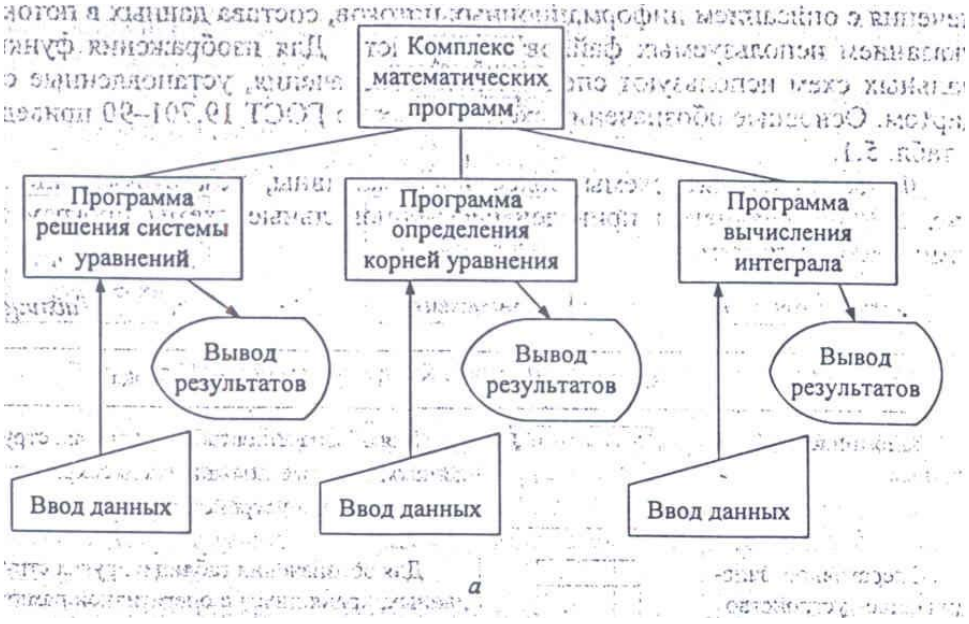
Рис. 5.2. Пример структурной схемы программной системы

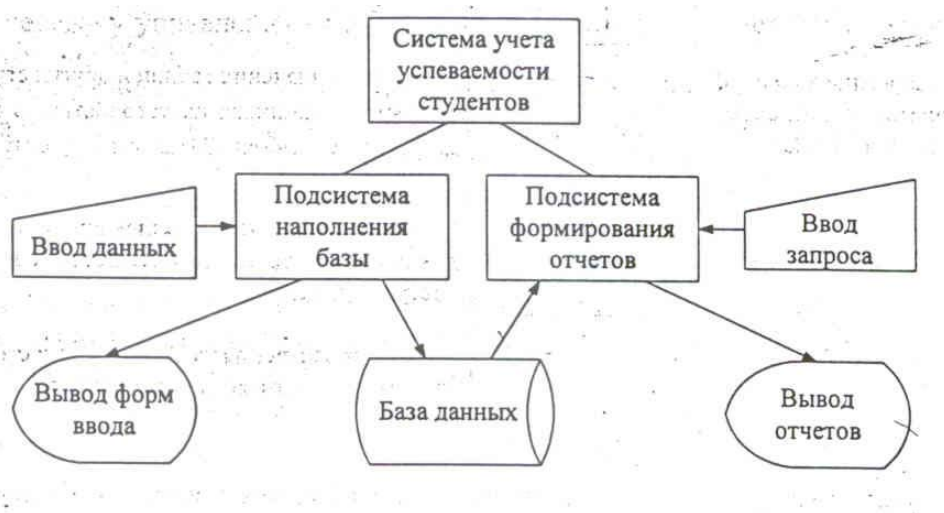
Более полное представление о проектируемом программном обеспечении с точки зрения взаимодействия его компонентов между собой и с внешней средой дает функциональная схема.

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. Для изображения функциональных схем используют специальные обозначения, установленные стандартом. Основные обозначения схем данных по ГОСТ 19.701-90 приведены в табл. 5.1.

Функциональные схемы, более информативны, чем структурные. На рис. 5.3 для сравнения приведены функциональные схемы программных комплексов и систем.

Все компоненты структурных и функциональных схем должны быть описаны. При структурном подходе особенно тщательно необходимо прорабатывать спецификации межпрограммных интерфейсов, так как от качества их описания зависит количество самых дорогостоящих ошибок. К самым дорогим относятся ошибки, обнаруживаемые при комплексном тестировании, так как для их устранения могут потребоваться серьезные изменения уже отлаженных текстов.





б

Рис. 5.3. Примеры функциональных схем:
а – комплекс программ; *б* – программная система

Таблица 5.1

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

5.2. Использование метода пошаговой детализации для проектирования структуры программного обеспечения

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 (см. § 1.3) реализует нисходящий подход (см. § 2.3) и базируется на основных конструкциях структурного программирования (см. § 2.4). 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 относительно 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 5.1. 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. 5.4, а), выполненного в некотором масштабе, в соответствующее изображение в окне на экране (рис. 5.4, б).

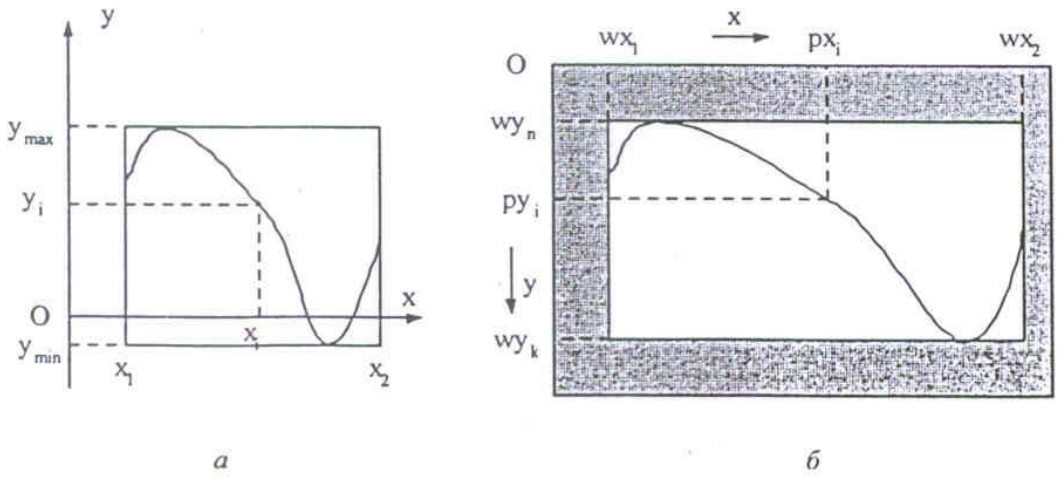


Рис. 5.4. Задача построения графика функции:
а – график функции; *б* – его отображение в окне на экране

Для построения графика необходимо определить масштабы по осям координат:

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

и координаты точек графика:

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

$$py_i = \lceil (y_{\max} - y_i) * m_y \rceil + wy_1.$$

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

$$Ipm_x = \frac{wx_2 - wx_1}{nI_x}, \qquad Ipm_y = \frac{wy_2 - wy_1}{nI_y},$$

где nI_x, nI_y - соответственно количество вертикальных и горизонтальных линий.

Для разметки сетки необходимо определить шаги разметки по горизонтали и вертикали:

$$\mathrm{Im}_x = \frac{x_2 - x_1}{nI_x}, \qquad \mathrm{Im}_y = \frac{y_{\max} - y_{\min}}{nI_y}.$$

Таким образом, для того 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: wx_1, wy_1, wx_2, wy_2 и количество линий сетки по горизонтали и вертикали nl_x, nl_y. Значения $wx_1, wy_1, wx_2, wy_2, nI_x, nI_y$ 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 (см. рис. 8.5). Для каждого пункта этого меню необходимо реализовать сценарий, предусмотренный в техническом задании.

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

Программа.
Инициализировать глобальные значения
Вывести заголовок и меню
Выполнять
Если *выбрана Команда*
то *Выполнить Команду*
иначе *Обработать нажатие клавиш 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 xl,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
Вее-выбор

Определим, какие фрагменты имеет смысл реализовать в виде подпрограмм. Во-первых, фрагмент Вывод заголовка и меню, так как это достаточно длинная линейная последовательность

операторов и ее 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 - подпрограммы первого уровня, которые определяют структуру программы (рис. 5.5). Определим для этих подпрограмм интерфейсы по данным с основной программой, т. е. списки параметров.

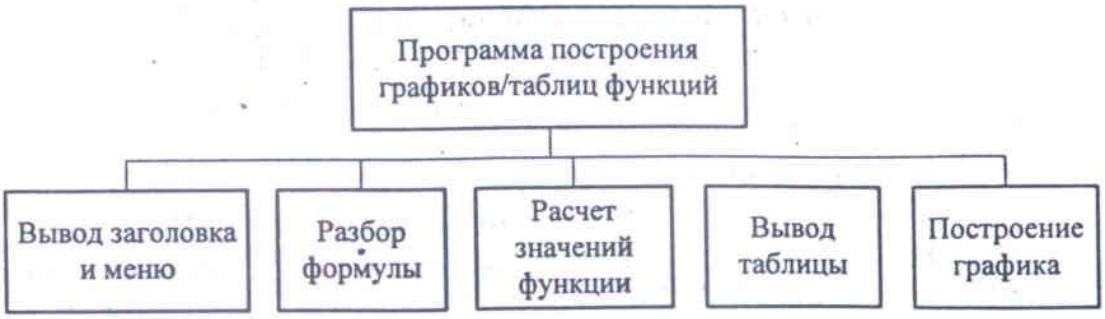


Рис. 5.5. Структурная схема программы построения графиков/таблиц функций.

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 будет выглядеть следующим образом:

Программа.
Вывод заголовка и меню
Выполнять
Если *выбрана Команда*
то
Выбор *Команда*
Функция:
Ввести или выбрать формулу Fun
Разбор формулы (Fat; Var Tree)
Отрезок:
Ввести значения xl,x2
Шаг:
Ввести значения h
Вид результата:
Ввести вид_результата
Выполнить:
Расчет значений функции (xl, x2, h, Tree; Var X,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. 5.6.

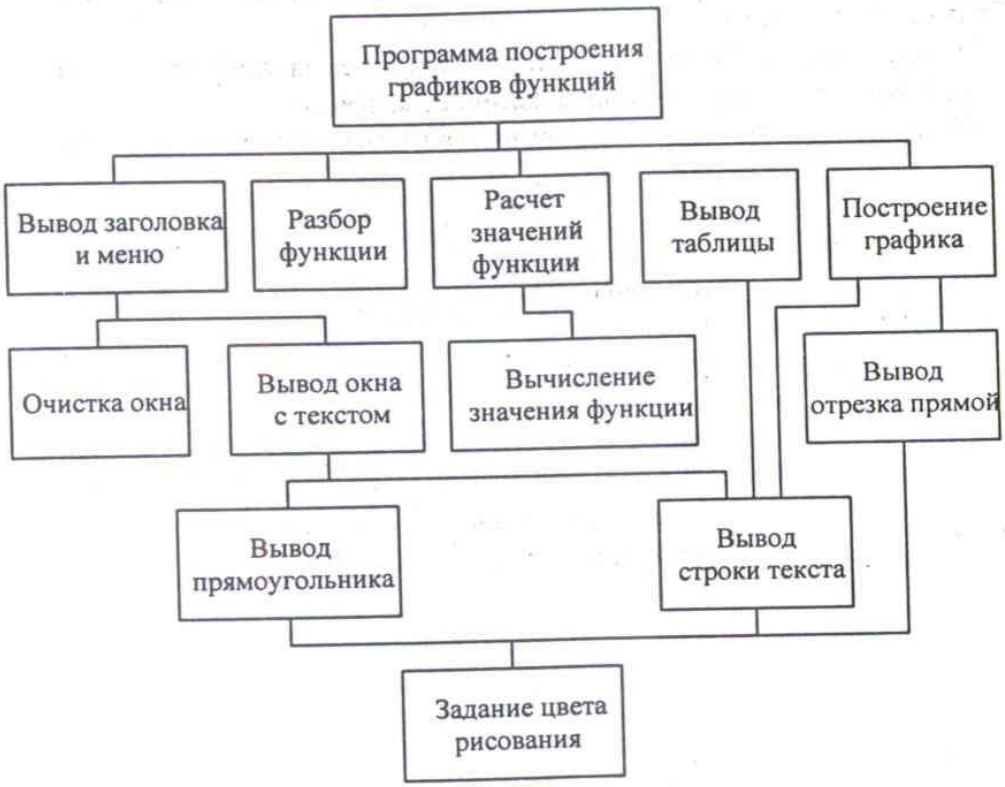


Рис. 5.6. Полная многоуровневая структурная схема программы построения графиков/таблиц

Как уже упоминалось в § 2.4, использование метода пошаговой детализации обеспечивает высокий уровень технологичности разрабатываемого программного обеспечения, 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 принципе 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, рассмотренные в § 2.2.

Для анализа технологичности полученной иерархии модулей целесообразно использовать структурные карты Константайна или Джексона.